ON THE PROCESSING OF COMPRESSED AND ENCRYPTED SIGNALS AND IMAGERY, WITH APPLICATIONS IN EFFICIENT, SECURE COMPUTATION

By

MARK STEVEN SCHMALZ

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1996

Copyright © 1996 by Mark S. Schmalz

ACKNOWLEDGMENTS

Many thanks are given to my family, for their loving support and encouragement, and to Drs. Ritter, Wilson, Laine, Rajasekaran, and Harris for their advisement of this work and associated publication efforts. Special gratitude goes to Gerhard Ritter, for his patient advice and encouragment in the development of unifying theory.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS iii

LIST OF FIGURES vi

LIST OF TABLES viii

ABSTRACT ix

1. INTRODUCTION 1
1.1. Study Overview 2
1.2. Previous Work 4
1.3. Technical Approach 31
1.4. Novel Claims 32
1.5. Implementational Advantages and Disadvantages 33

2. REVIEW OF NOTATION 35
2.1. Overview of the Image Algebra (IA) Subset 35
2.2. Study Notation 46

3. FUNDAMENTAL THEORY 49
3.1. Mathematical Concepts. 49
3.2. Image Compression, Encryption, and Compressive Computation. 56
3.3. Taxonomy of Image Transformations. 73
3.4. Class-Specific Derivational Techniques. 79

4. COMPRESSIVE PROCESSING — COMPUTATIONAL COMPLEXITY AND DATA SECURITY 117
4.1. Complexity of Image Algebra Operations. 118
4.2. Complexity of Compressive Computation. 124
4.3. Feasibility of Compressive Computation. 134
4.4. Data Security. 141
4.5. Feasibility of Encryptive Computation. 158

5. COMPUTATIONAL ERROR AND INFORMATION LOSS 164
5.1. Theory of Error Propagation in Discrete Systems 164
5.2. Error Propagation in Discrete Image Algebra Operations 173
5.3. Theory of Error-Tolerant Computation 180
5.4. Feasibility of Error-Tolerant Computation 182
5.5. Information Theory and Error-Tolerant Computation. 187

6. CLASS 1 TRANSFORMATIONS 198
6.1. Substitutional Ciphers. 198
6.2. Transpositional Ciphers. 208
6.3. Linear Transforms. 211

7. CLASS 2 TRANSFORMATIONS 223
7.1. Generalized Class 2 Transform. 223
7.2. Pixel Value Compression via Bit Slicing. 228

8. CLASS 3 TRANSFORMATIONS 236
8.1. Affine Transformation. 236
8.2. Spatial Transformation by Pixel Selection 247
8.3. Transpositional Ciphers. 253

9. CLASS 4 TRANSFORMATIONS 259
9.1. Block Encoding. 259
9.2. Sparse Matrix Processing. 261
9.3. Transform Coding. 265
9.4. JPEG. 268
9.5. Block Truncation Coding and VPIC. 276
9.6. Vector Quantization. 291

10. APPLICATIONS OF COMPRESSIVE IMAGE PROCESSING. 302
10.1. Compressive Smoothing 302
10.2. Compressive Edge Detection. 310
10.3. VPIC Morphological Operations. 323
10.4. High-Level Compressive Computations. 326

11. APPLICATIONS IN PARALLEL COMPUTING 334
11.1. Effect of Domain Compression Ratio. 334
11.2. Effect of Range Compression Ratio. 338
11.3. Simplification of Operations. 339
11.4. Partitioning Efficiency. 341

12. CONCLUSIONS 346
12.1. Conclusions. 346
12.2. Open Issues and Future Work 350

REFERENCES 356

BIOGRAPHICAL SKETCH 362

LIST OF FIGURES

1. Commutativity diagram for processing compressed or encrypted imagery with unary operations. 59

2. (a) Example source image a that is transformed by T to yield (b) image a_c. Note reversal of the middle 4×4-pixel neighborhood only. 82

3. Example of block fragmentation in pointwise compressive operations over non-identically tesselated domains: (a) representation of the y-th block in a (solid line) and b (dotted line), (b) resultant block decimation required to process the compressed images $\mathbf{c_c} = \mathbf{a_c} \bigcirc' \mathbf{b_c}$ such that corresponding sub-blocks are amenable to pointwise combination. 113

4. Commutativity diagram of a compressive computational system with multiple encoding transforms T_i and multiple analogues $\bigcirc_i'$ of image operation $\bigcirc$. 136

5. Commutativity diagram of a compressive computational system with multiple encoding transforms T_i, $i \in \mathbf{Z}_\zeta$, and multiple analogues $\bigcirc_{j,i}'$, $j \in \mathbf{Z}_\psi$ of the j-th image operation $\bigcirc_j$, where $\zeta = 2$ and $\psi = 2$. 137

6. Information-theoretic model of communication. 187

7. Information-theoretic model of communication. 187

8. Arrangement of noise levels in a signal a with entropy H(a) that partially occupies a channel of bandwidth B having K noise bits. 195

9. Local averaging of a noisy, two-dimensional, 8-bit image transformed by $T(x) = (x+128) \bmod 256$: (a-b) source image and template a and t; (c-d) transformed image and template a_c and s; (e) transformed result b_c; (f) image-domain result b $= T^{-1}(b_c)$. 206

10. Dual of pointwise multiplication over the range space of the pointwise linear transform $T(x) = cx + d$. 214

11. Recovery of source image a from image sum of contrast-stretched imagery: (a-b) source images a and b, (c-d) linear contrast-stretched images $a_c = T(a)$ and $b_c = T(b)$, (e) $c_c = a_c$ and b_c, (f) recovery of a from c_c using $a = T^{-1}(c_c - T(b))$. . . 215

12. Pixel indexing and overlap scheme for information loss analysis of an affine transform (a) indexing scheme and (b) example of pixel overlap. 238

13. Local averaging of a rotated Boolean bar chart using a rotated template: (a-b) source image a and template t; (c-d) rotated image $a_c = T(a)$ and template s formed by applying T to the weight matrix of t, then restricting to the 3×3-pixel Moore configuration; (e) rotated locally-averaged image $a_c \oplus s$; (f) derotated image $T^*(a_c \oplus s)$. 251

14. Error images of local averaging using rotated and unrotated templates: (a-b) error images e (Equation 307) and f (Equation 308), taken from the central (uncropped) portion of domain(a); (c-d) histograms of error images e and f. . . 252

15. Error and efficiency measures associated with JPEG addition over natural scenes having 3 bits/pixel error. 275

16. Codebook growth and error as a function of codebook size and Methods 1–3 for VQ pointwise addition: (a) input and output code book sizes M and N, (b) input and output errors ϵ_i and ϵ_o. 300

17. Block configuration for compressive local averaging, showing block neighborhoods (dashed boxes) on X returned by c, r, and j. 304

18. Example of compressive smoothing at $CR_d = 16{:}1$: (a) source image, (b) locally averaged image using unitary von Neumann template, (c) compressed array of block means over which processing actually occurs, (d) decompressed result of compressive smoothing. 307

19. Example of VPIC coding of very low-resolution Boolean imagery: (a) source image, (b) VPIC representation of a), where M,x denotes a mean block of mean x. . . 318

20. Example of VPIC coding of very low-resolution Boolean imagery: (a) boundary-detection of Figure 19a), (d) VPIC edge detection of Figure 19a). . . 319

21. Error analysis of the edge detector in Figure 20b, in terms of erroneous source pixels per encoding block. 320

22. Greylevel edge detection with VPIC: (a) source image, (b) Sobel edge detection, (c) 4×4-pixel VPIC edge detection using the codebook similar to that given in Example 10.2.2.3, (d) noise and representational error as a function of VPIC blocksize (k×k pixels) for underwater and land-based imagery. 321

23. Target characterization using VPIC codebook exemplars with indices in {1,2,3}: (a) VPIC codebook, (b) source image with unitarily-valued target region, (c) compressed representation, where 1′ denotes exemplar 1 rotated by 90 degrees. . 328

24. Example of VPIC-based target recognition: (a) source image, (b) BTC-compressed image over a portion of which the target recognition algorithm (Equation 410) computes, (c) decompressed target location image. 330

25. Conversion of a SIMD-parallel mesh to pipelined computation using compressive processing: (a) input and compute over compressed image 1, (b) input compressed image 2 and compute over image 1, (c) input compressed image 3 and compute over images 1 and 2, (d) input compressed image 4, compute over compressed images 1–3, and output compressed image 1. 337

LIST OF TABLES

1. Costs involved in SIMD-parallel computation of a 2–pixel image-template convolution versus a 5–pixel LUT operation over VPIC-format imagery . . 339

2. Processor cycles incurred by SIMD-parallel computation of a 2–pixel image-template convolution versus a 5–pixel LUT operation over VPIC-format imagery 340

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

ON THE PROCESSING OF COMPRESSED AND ENCRYPTED SIGNALS AND IMAGERY, WITH APPLICATIONS IN EFFICIENT, SECURE COMPUTATION

By

Mark Steven Schmalz

August 1996

Chairman: Gerhard X. Ritter
Major Department: Computer and Information Science and Engineering

In this dissertation, we develop theory and analyses pertinent to the processing of compressed or encrypted imagery, called *compressive processing* or *encryptive processing*. Encryptive processing, which is a long-standing goal of computer science, exploits the processing of secure (encrypted) operands to yield a secure computation. Unfortunately, development has been hindered by an obscure understanding of mathematical concepts fundamental to the encryption process. Similarly, the processing of compressed imagery can yield computational efficiencies as a result of fewer input data. However, such speedups are not uniform across commonly-used image compression formats and may not exist for certain operations over the range spaces of given transforms. Additionally, the formulation of compressive transforms can be quite involved, which intuitively suggests that the derivation of compressive computational operations is difficult in closed form.

We first discuss theory that is basic to an understanding of compressive and encryptive processing. Using image algebra, a rigorous set-theoretic and functional mapping notation,

we derive novel algorithms for implementing selected pointwise arithmetic, neighborhood-, and domain-specific operations on transformed data. Our algorithms are both feasible and portable to numerous computers, since image algebra has been implemented on a variety of serial and parallel processors, including the Connection Machine, MasPar SIMD processor, ERIM's Cytocomputer, Alliant Tech's PREP, Inmos' Transputer architecture, and the Martin-Marietta GAPP-IV processor upon which the PAL architecture is based.

Subsequent implementational discussion emphasizes the feasibility of sequential and parallel compressive computation. We show that compressive processing methodologies can be mapped to a variety of well-known architectures, especially SIMD array processors. Analyses focus upon time complexity, cost, and error due to information loss. For example, we show that compressive processing can reduce the processor count in parallel architectures without compromising the computational speedup obtained through parallelism. A further advantage of selected compressive transforms is their ability to facilitate the mapping of costly algorithms such as edge detection and component labelling to compressive operations implemented in terms of lookup tables. Such techniques require only I/O operations and can be stored in local memory in certain SIMD-parallel processors.

CHAPTER 1
INTRODUCTION

The progress of image and signal processing research has long been hindered by the insufficient computational bandwidth of available computing machinery. Due to the large data burden presented by high-resolution imagery, near-real time processing of large images (e.g., surveillance or medical imagery) is especially costly. Recent research [1–6] has explored the possibility of processing compressed imagery, based on the observation that compressed images have fewer data, as well as the conjecture that fewer input data require fewer operations. In practice, we have found that certain image processing operations can be mapped to the range spaces of specific transformations, such that decompressing the processed compressed image yields an exact representation to (or approximation of) the processed source image. We call such transform-regime operations *analogues* of the corresponding image operations over the given transform's range space. When one computes an image operation or sequence of such operations using compressed imagery, that is called *compressive processing*. In practice, it is possible that certain formulations of analogous operations can yield a computational speedup since: (a) the analogous operations may be more efficient than the corresponding image operation, or (b) fewer data may be processed by such operations.

Similar to the concept that computational speedup could accrue from the processing of compressed imagery, it has been observed that data security could be maintained throughout a computational system by appropriately processing encrypted data. In contrast to compressive transforms, which are extensively discussed in the literature, the formulation of numerous practical encryptive transforms remains obscure. Additionally, we note that

many compressive transforms can be thought of as having weak encryption capabilities, due to the visual obscurity of compressed data formats.

1.1. Study Overview

In this dissertation, we address the following key questions:

1. Can compressive or encryptive transforms be classified in a manner that facilitates convenient derivation of analogues to common image processing operations?
2. Given the aforementioned classification scheme and a particular transform class, can methods for deriving analogues over the range spaces of multiple transforms in that class (or a relevant subclass) be elucidated? If not, then why not? If so, then how can one employ such methods effectively, and to what types of image operations or operands do they apply?
3. Given the presence of noise and computational error in imagery and image algorithms, respectively, what is the noise sensitivity of compressive processing? This is an important implementational issue, since image compression decreases redundancy in the compressed image. In the source image, noise in a given pixel is generally localized to the pixel's domain point. However, due to reduced redundancy in the compressed image, noise in a givenpixel value of a compressed image could cause perturbation of multiple pixel values in the corresponding decompressed image.
4. Assuming that we can derive an analogue of an image operation over the range space of a compressive transform, what computational advantages can be obtained via image processing with such analogous operations? Additionally, is it possible to cascade compressive operations, thus facilitating algorithm development in the transform domain via composition of known analogues?

5. What additional insights, effects, or implementational advantages could accrue from compressive or encryptive processing? How would such concepts be useful in computer science?

Given the preceding questions, we conclude this chapter with a brief perspective on the small amount of previous work reported in the literature, as well as comments on methods and potential advantages of our approach. Chapter 2 begins with an overview of notation and Chapters 3–5 summarize theory that supports the derivation of analogues of image processing operations. We choose several commonly-employed transforms from each class of our transform taxonomy. Given a transform T, we discuss the derivation of selected pointwise, global-reduce, and image-template operations over *range*(T). In certain cases, we have discovered methods for deriving analogous operations that are applicable to a large subclass of transforms in the given class.

The latter portion of this dissertation contains an implementational discussion that emphasizes three topics. First, we consider the effects of compressive image processing with three specific transforms (JPEG [7], Vector Quantization [8], and Visual Pattern Image Coding [9]). Second, we analyze the propagation of error through a discrete compressive processing system, in terms of the communications and computational errors manifested by given operations over the range space of each transform. Third, we discuss problems and pitfalls related to the implementation of such analogous operations on sequential and parallel processors. Implementational issues include space complexity as well as limits on computational and communications bandwidth. In particular, we discuss the reduction in parallelism that may be achieved via compressive processing on SIMD-parallel meshes. Such effects pertain primarily to medical and military applications in real-time image processing at high frame rates.

Conclusions and suggestions for future work emphasize the impact upon image processing that compressive computation may produce, under the assumption that error propaga-

tion can be predicted and controlled. An additional assumption pertains to the availability of analogous operations over the range spaces of compressive transformations that may be obtained from future development efforts. Open issues, such as the elucidation of general derivational methods for analogues over a transform class or subclass, are also considered.

1.2. Previous Work

The computational advantages of processing fewer data were long ago realized in the processing of reduced matrices [10,11]. Additionally, electrical engineering practice has long emphasized operations over parameter spaces that approximately characterize input signals. Such feature-space approximations usually implement one or more forms of data reduction, but can require exotic algorithms or architectures for computing simple operations [12]. Although of implementational interest to certain mission-specific efforts, such feature space operations generally exhibit disadvantages of high space complexity as well as costly hardware and software development. Thus, such methods are not generally attractive for commercial applications. As a result, additional research is required in the processing of compressed data.

In order to acquaint the reader with the scope of this study and provide salient detail, we begin our literature review with a discussion of transforms that have been developed specifically for the purpose of data compression (Section 1.2.1). This overview supports the primary topic of this study, namely, compressive processing. We then review data encryption (Section 1.2.2), which provides background for a summary of encrypted data processing (Section 1.2.3). Such background leads naturally to a discussion of processing compressed signals and imagery, which is given in Section 1.2.4.

1.2.1. Data Compression

Research endeavor in signal and image compression has been extensive, due primarily to the economic benefits of exploiting communication channel bandwidth. In principle, compression is achieved by reducing redundancy in the source data. For example, an image of constant value k can be represented by one number (k), regardless of image size. In contrast, an image consisting of randomly chosen pixel values cannot be compressed. Early on, it was recognized that the first derivatives of many signals exhibited lower information content than the source signal. This lead to the development of primitive methods of image compression such as *delta modulation* [13] and *differential pulse code modulation* (DPCM). As a consequence of the need for accurate, efficient quantizers in PCM and DPCM, a variety of statistical compression techniques were developed including *adaptive DPCM*, which were based upon adaptive or recursive quantization techniques [14].

As digital images increased in size and thus required higher channel bandwidth and increased storage, data compression research began to emphasize two-dimensional transformations. For example, the concept of subdividing an image into blocks (generally of rectangular shape) gained popularity due in part to the limited memory then available on fast signal processors. Such methods, called *block encoding* (BE), tessellate the image domain into encoding blocks that exhibit greylevel configurations which can be represented in lossless form (e.g., via indexing) or with a lossy transform such as *vector quantization*. Since there are fewer block configurations than there are possible blocks, and a group of values are represented by an index or by an *exemplar block* (also called a *vector*), VQ produces the desired effect of image compression. By arranging the input data to achieve maximum intra-vector and minimum inter-vector correlation, the compression ratio of VQ can be increased, which partially offsets the effort required by the determination of the VQ *codebook* (a set of exemplar vectors) [15].

An alternative method of block encoding, called *block truncation coding* (BTC) [16], encodes regions of low greylevel variance in terms of a mean value and regions of high variance in terms of a mean, standard deviation, and a residual bitmap that denotes the positions of zero crossings. Unfortunately, BTC is expensive computationally, due to the adjustment of bits in the bitmap to effect reduced entropy. Since the cost of BTC increases exponentially with blocksize [9,16], and BTC has a compression ratio that is moderate by today's standards (typically 20:1 to 25:1 for images of natural scenes), BTC is not considered a compression transform of choice.

Additionally, by transforming the output of a block encoder with, for example, the *Fourier* or *Cosine transforms*, then selecting transform coefficients that are deemed significant a priori, one can further reduce the image data burden, although at the cost of information loss. Such methods are generally called *transform coding* [17], and feature prominently in compression schemes (e.g., JPEG, MPEG) that are currently in vogue for digital telephony applications [18]. By following the coefficient selection stage (i.e., *quantization step*) with a provably optimal, lossless compression transform such as *Huffman encoding* [19], one can obtain further data reduction and thus achieve higher compression ratios without incurring information loss.

More exotic methods of image coding are based upon the reduction of an image to significant eigenvalues, as in the *Singular Value Decomposition* (SVD) [20]. Although proven optimal for image compression, eigenvalue transforms such as the SVD and *Karhunen-Loeve transform* (KLT) are computationally burdensome, requiring $\mathbf{O}(n^4)$ work for an n×n-pixel image. Thus, the SVD and KLT are infrequently employed, despite the ease with which significant transform coefficients (eigenvalues) may be obtained from the transformed image.

An alternative method of compression by *recursive decomposition*, which is often based upon knowledge derived from observation of the human visual system (HVS), has been employed with some success but tends to be data-dependent. For example, early attempts at multifrequency decomposition, such as the *Synthetic High* technique [17], eventually

led to Fourier-transform based methods currently known as *wavelet based compression* [21]. Similarly, Barnsley et al. [22,23] have published extensively in the area of image compression based on recursive, VQ-like decompositions that derive their methodology from the *collage transform* and from concepts of *fractal geometry* [24]. Due to an obscure formulation and high cost of the forward transformation, fractal-based compression remains an experimental technique.

Recently-published research by Chen and Bovik [9] has reconsidered the problem of compression based upon HVS-based knowledge of exemplar patterns that may be used by the visual cortex to partition imagery. For example, the visual cortex is known to contain simple, complex, and hypercomplex cells [25] that mediate receptive fields which detect bar- or wedge-shaped gradients, as well as feature orientation and (possibly) shape. Chen has exploited this information to modify block truncation coding by replacing the costly bitmap construction step with a simple correlation step that finds the best match between the zero crossing configuration and a small set of exemplar blocks. This method, called *visual pattern image coding* (VPIC) yields high computational efficiency with respect to BTC or VQ and combines advantages of both methods. Given the appropriate set of exemplar patterns, Chen has demonstrated (on several standard images) high reconstruction fidelity at compression ratios of 20:1 to 40:1, which appears to be superior to JPEG's performance at such compression levels.

We next consider the allied topic of data encryption, from which area of study signal and image compression theory arose.

1.2.2. Data Encryption

The development of data encryption transforms has long been of interest due to applications in military science and statecraft [26]. More recently, business communications research has shown great interest in data encryption, as applied to the security of financial

transactions. Beginning with *monoalphabetic substitutions* and *transpositional ciphers* [27], data encryption progressed to *anagrammatic ciphers*, as well as *polyalphabetic encryptions* based on the *Vigenère cipher* [28]. The vulnerability of such methods to attack based upon the ciphertext histogram and knowledge of plaintext statistics led to development of the *Vernam cipher* [29], which is based upon modulo-n addition and subtraction. The Vernam cipher, although used extensively during WWI, can be compromised if an intermediate result is detected, or if a portion of the plaintext is known. In response to this situation, development centered on *Hill's linear k-gram cipher* [27] and *rotor machines* that were sophisticated electro-mechanical implementations of the Vignère cipher.

The Hill cipher requires solution of a system of k linear equations, in which the alphabetic indices of k elements of plaintext are multiplied by an encryption matrix to yield ciphertext. The product of an inverse of the encryption matrix and the ciphertext produces the plaintext. Although the Hill cipher was initially thought to be secure, it was quickly proven that linear-algebraic attack could compromise the encryption in $O(k^3)$ time [27]. Likewise, rotor machines incur the basic deficiencies of the Vigenère cipher, which are: (a) a finite number of alphabets that is small relative to customary message size, (b) resulting susceptibility to attack based upon known plaintext statistics and semantics, (c) insecure key distribution schemes, and (d) vulnerability to automated (i.e., computational) attack. In the latter category was much pathfinding research in pattern matching and enumerative attack using *Turing machines*, as embodied in the Atlas computer employed by the British during WWII in their cryptology center at Bletchley Park. It this context, it is interesting to note that the rotor machines used by the Axis powers in WWII were descended from a machine that was patented in the US in the 1920s [30]. Such narrative is well documented in the literature [26], and are thus not considered further in this discussion.

Based upon cryptanalytic experience gained during WWII, Claude Shannon in 1949 published suggestions concerning features of cryptosystems that might be useful in the future, such as interleaving of substitutions and transpositions [30]. Throughout the

1960s, the IBM Corporation developed Shannon's ideas into a workable prototype cipher, called LUCIFER, which was improved in the early 1970s to yield the *Data Encryption Standard* [29]. In its original form, DES accepted a 56–bit Boolean key and a 64–bit Boolean plaintext, and returned a 64–bit Boolean ciphertext. In practice, the DES transformation consists of 18 component transformations that are composed to yield a sequence of substitution and transposition operations. The transpositions are based in part upon hitherto undisclosed functions called *S-boxes* that are implemented in publically-available lookup tables [30].

Due to the possibility of compromise via enumerative attack facilitated by increasingly powerful computers, it was proposed in the mid-1980s that DES' input be extended to 128 bits, and that the key be of similar size. Note that: (a) this extension has yet to be adopted, (b) the US Department of Defense (DoD) until recently continued to recommend the 58–bit DES keyspace as the standard for business communication, and (c) DES has not been employed by the DoD in military encryption. The latter situation was underscored by Diffie and Hellman's proposal [31] of an exhaustive attack on DES, in which a multiprocessor architecture enumerates the 2^{56} possible keys. Given the 1ns cycle time of simple, fast, commercially-available uniprocessors (GaAs or HEMT technology [32]), and assuming the existence of a 10^4-processor array, 10^{13} keys could be generated per second. Since DES is inherently pipelined, and $2^{56} \approx 7 \times 10^{16}$, then approximately 7000 seconds, or less than two hours, would be required for brute-force decryption. Additional methods of attacking DES include guessing of the key using techniques based upon genetic algorithms [33], which have proven efficient in attacks upon rotor machines [34].

From the preceding discussion, it should be obvious that there exist serious problems with cryptosystems that have only one key. In particular, extreme vulnerability occurs when the key is distributed to numerous communicating parties. For example, consider network communications, where a fully connected network with n nodes requires $\mathbf{O}(n^2)$ key

transmissions. Clearly, the probability of key seizure (or, at the very least, key monitoring) by unauthorized persons increases polynomially with the number of communicating parties.

For example, if N = 4, then there are 4^2–4 = 12 key transmissions required. However, if N = 8, then 56 keys are transmitted. If one assumes a network organized around a high-speed trunk or backbone line, then the second case (N=8) has a probability of key transmission per encoded message that exceeds 4.5 times the probability of key transmission when N=4.

As a partial solution to the key distribution problem, Diffie and Hellman [31] proposed that the key be partitioned into two halves, i.e., $\mathbf{k} = (\mathbf{k_p}, \mathbf{k_s})$ where the *public key* $\mathbf{k_p}$ is the only key required for *encryption*, but $\mathbf{k_s}$ is required for *decryption*. Implementationally, let the i-th user in an N-user system choose key $\mathbf{k_i} = (\mathbf{k_p}(i), \mathbf{k_s}(i))$, and let a directory readable by all users list the public keys $\mathbf{k_p}(i)$, i = 1..N. Let a plaintext message **a** be sent to user i as ciphertext $\mathbf{a_c}$, which is defined as

$$\mathbf{a_c} = T(\mathbf{a}, \mathbf{k_p}(i)), \tag{1}$$

where T: $\mathbf{F^X} \times \mathbf{K} \rightarrow \mathbf{F^X}$ denotes an encryption transform. Only user i has the secret portion $\mathbf{k_s}(i)$ that is required to compute the decryption T^*: $\mathbf{F^X} \times \mathbf{K^2} \rightarrow \mathbf{F^X}$ as

$$\mathbf{a} = T^*(\mathbf{a_c}, [\mathbf{k_p}(i), \mathbf{k_s}(i)]). \tag{2}$$

Thus, in public-key systems, only keys $\mathbf{k_p}(i)$, i = 1..N, are distributed openly, versus the secret distribution of $\mathbf{O}(N^2)$ keys required by private-key cryptosystems. Additionally, *public-key cryptography* (PKC) can be used to authenticate transmissions when a message is accompanied by an encrypted *signature*, which may be appended automatically. The discussion of authentication, which is beyond the scope of this dissertation, is summarized in Reference 29.

Although attractive conceptually, PKC (Public Key Cryptography) has two stringent requirements, which can be stated in terms of the following related problems:

a) Given a sum of two numbers, determine the addend and augend; and

b) Given a product of two numbers, determine the multiplicand and the multiplier.

In practice, such requirements have been fulfilled by the following cryptosystems.

Merkle and Hellman's *knapsack cryptosystem* [35] was the first public-key system proposed in the open literature. Concurrently, Graham and Shamir developed and later published [36] a slightly different approach to knapsack systems, which is distinguished by the construction of *knapsack sets* and by a modification to Merkle and Hellman's decryption step. Another approach to knapsack systems is the *iterated knapsack*, which uses multiple knapsack sets, similar in concept to the polyalphabetic substitution that employs multiple alphabets. Knapsack sets are often implemented in terms of *superincreasing sets*, which can be constructed from an indexed set $\mathbf{F}$, as follows

$$\mathbf{F}_{si} = \left\{ f_i \in \mathbf{N} : f_i > \sum_{j=1}^{i-1} f_j \right\} \subset \mathbf{F}. \tag{3}$$

An unfortunate property of superincreasing sets, which defeats the security of knapsack encryption, is that the ratio of the number of elements to the number of bits required to encode the set is not greater than 1. The greater the radix r of a superincreasing set $\mathbf{F}_{si} = \{r^i \in \mathbf{R} : r > 1 \text{ and } i \in \mathbf{N} \cup \{0\}\}$, the further apart are such numbers on the number line. Unfortunately, such *low-density sets* yield knapsack encryptions that can be easily broken. However, by constructing *hard knapsack sets*, which are denser and yield encryptions that are difficult to break, it has been shown that a successful computational attack is at least NP-complete [30]. Methods of attack on knapsack systems have been proposed by Shamir [37], Lagarias and Odlyzko [38], as well as Brickell [39]. A summary of such methods is beyond the scope of this overview, but can be found in References 29 and 30 , which review knapsack systems in detail.

Rivest, Shamir, and Adelman (*RSA*) proposed [40] that public-key cryptography be implemented in terms of the product of prime numbers. Their system currently remains secure, i.e., no successful comprehensive attack on RSA has been reported in the open literature. However, the RSA algorithm, as well as a feasible RSA decryption scheme, has been disclosed in the *Pretty Good Privacy* (PGP) encryption algorithm which was made publicly available in 1994. As a result, RSA is generally considered insecure for purposes of military communications.

With the development of efficient prime factoring algorithms, as well as the storage of precomputed prime factors, RSA has become susceptible to attack unless the number of digits k is very large. Furthermore, the processing of data encrypted by RSA requires decryption for numerous operations to be feasible implementationally. Although multiplication can be performed upon the encrypted result, the general equation $y = a + bx$ cannot be solved for y given x, since RSA is based upon the multiplication of exponents. Thus, the utility of RSA in processing encrypted data appears to be limited to integral scaling problems.

Additionally, efficient implementation of the RSA algorithm requires reasonably fast algorithms for determining numbers that are relatively prime in a system based upon modular arithmetic. Furthermore, RSA requires the efficient computation of inverses in a modular system, as well as the computation of large exponents. Such algorithms have been published recently [41].

Methods of encryption that are currently in vogue exploit the geometry and nonlinearity of certain bivariate functions (such as conic sections) to produce two keys that satisfy constraints of the PKC scheme. Such methods yield a large search space, but can be compromised if the geometric figure upon which they are based can be guessed or approximated computationally [29]. An analysis of such implementational techniques is beyond the scope of this dissertation, but may be found in Reference 29.

1.2.3. Encryptive Processing

The processing of encrypted data, which we call *encryptive processing*, predated research in compressive processing by nearly two decades. Encryptive processing seeks to mathematically combine ciphertext and plaintext data in a manner that is secure from the site of encryption to the site where the data is combined, and remains secure to the decryption site. Such methods are useful for financial transactions, updating of personal (i.e., medical) records without compromising privacy, and processing of classified data such as military surveillance imagery. Our review of previously-published work in encryptive processing begins with Rivest's, Adelman's, and Dertouzos' concept [42] of linear homomorphisms, which have been shown to be vulnerable to linear-algebraic attack, but remain interesting pedagogically.

Additionally, we overview analyses by Abadi et al. [43] of the likelihood of encrypting polynomial-time and NP-complete functions, and discuss recent work by Ahituv [44]. The latter research is based upon the assumed combinatoric complexity of recursive encryption, and provides an in-depth analysis of cryptographic vulnerability in practical encryptive computational scenarios. We also consider binary superposition operations over plaintext-ciphertext pairs formulated by Yu and Yu [45], which are feasible in practice, due primarily to an efficient formulation, deep recursive encryption, and large plaintext size. Although an intermediate result may be discovered, its decryption can be rendered infeasible in operational scenarios due to practical constraints upon an adversary's ability to enumerate the recursive encryption steps.

1.2.3.1. Linear privacy homomorphisms. Difficulties in processing encrypted data were recognized early on, and were discussed in Rivest et al.'s paper [42] of 1978. However, linear homomorphisms can be successfully attacked via the solution of a set of linear equations, as noted in Reference 44.

Given spaces S and S′, as well as functions $f : S \rightarrow S$ and $f' : S' \rightarrow S'$, the transformation $T : S \rightarrow S'$ is a homomorphism if and only if $f'(T(g)) = T(f(g)), g \in S$. Additionally, if T is a linear transform, as well as a homomorphism, then T is called a *linear homomorphism.*

For example, let **a** and **b** be n-element vectors over the alphabet **F**, and let T be a linear homomorphism. We thus have

$$T(\mathbf{a} + \mathbf{b}) = T(\mathbf{a}) + T(\mathbf{b}) \tag{4}$$

$$T(\mathrm{m}\cdot\mathbf{a}) = \mathrm{m}\cdot T(\mathbf{a}) = T(\mathbf{a})\cdot \mathrm{m}\,, \tag{5}$$

where m is a constant in **F**.

Let encrypted data be denoted by $\mathbf{a}_c = T(\mathbf{a})$ and $\mathbf{b}_c = T(\mathbf{b})$. Assuming that T has an inverse $T^{-1} : S' \rightarrow S$, one updates $\mathbf{a}_c$ with plaintext **c** as follows

$$\begin{aligned} &\text{STEP 1 : Encryption :} && \mathbf{c}_c = T(\mathbf{c}) \\ &\text{STEP 2 : Updating :} && \mathbf{a}_c^{\mathrm{new}} = \mathbf{a}_c + \mathbf{c}_c \\ &\text{STEP 3 : Decryption :} && \mathbf{a}^{\mathrm{new}} = T^{-1}(\mathbf{a}_c^{\mathrm{new}})\,. \end{aligned} \tag{6}$$

Multiplication by a constant is similarly performed, per Equation 5.

Linear homomorphic encryption systems are easily broken by solving n linear equations, as follows. Initially, one constructs a dictionary **d** consisting of n words. In a representation such as the binary number system, where the k-th bit of the k-th word is unitary and a vector $\mathbf{d}_k$ is zero elsewhere, we have

$$\begin{aligned} \mathbf{d}_1 &= T(1,0,0,\ldots,0) \\ \mathbf{d}_2 &= T(0,1,0,\ldots,0) \\ &\vdots \\ \mathbf{d}_n &= T(0,0,0,\ldots,1) \end{aligned} \tag{7}$$

In practice, an n-bit encrypted block can be defined as the sum of rows of **d**, due to the linearity property of T. For example, if we choose an enciphered message

$\mathbf{a}_c = (0,1,1,0,0,1,0,\ldots,0) \in \{0,1\}^n$ we have the linear equation $\mathbf{a}_c = \mathbf{d}_2 + \mathbf{d}_3 + \mathbf{d}_6$. or some linear combination of the rows of $\mathbf{d}$. By defining n such messages, one obtains n linear equations. By knowing the solution (i.e., the plaintext arguments of T that produced $\mathbf{d}$), the homomorphism can be broken via solving n linear equations in the customary manner.

Implementationally, given the $O(n^3)$ multiplication overhead of matrix inversion, 1024-byte signals (n=8192 bits, or 16 lines of 64 characters) could be inverted (with machines capable of 1 billion multiplies per second) in approximately 550 seconds = 8192^3 multiplications/10^9 multiplications/sec. As a result, such messages are not immune from attack with current computational hardware when the useful lifespan of a message is as short as several minutes. Thus, privacy homomorphisms remain of pedagogic interest primarily.

1.2.3.2. Abadi's analysis of encryptive processing. Given the apparent failure to secure simple operations such as addition and multiplication by a constant, one would naturally ask what functions are encryptable, and what level of security can thus be attained. Abadi, Fiegenbaum, and Kilian [43] considered the problem of encryptive processing in terms of the following scenario, which is based upon an *encryptable function f*:

1. Party A knows a datum x in $domain(f)$, but has insufficient resources to compute $f(x)$.
2. As a result, A requests from party B the quantity $f(y)$, where $y \in domain(f)$ and $y \neq x$. Since B will compute the function f that is hard from A's perspective, B can successfully attack a cryptosystem based upon intractability assumptions. However, B may not guarantee the privacy of the encrypted data.
3. Via an encryption, A (with inferior resources) can request $f(y)$ from B, but B cannot infer x from y.
4. However, A can infer $f(x)$ from $f(y)$, and does so, to obtain the desired result.

From item 3), one (correctly) concludes that A must be able to understand (and thus compromise) an encryption system that B cannot successfully attack. Therefore, A has *greater encryption resources* than B, but has *fewer computational resources* (as noted in Item 1). Via a series of interesting theorems, the authors claim to

- Prove precise statements about what information the computational operation f hides and discloses, in the information-theoretic sense;
- Develop theory that proves an encryption $T: x \vdash\!\!\rightarrow y$ hides or conceals at least the properties $\mathcal{H}(x)$ and discloses at least $\mathcal{D}(x)$;
- Give examples of natural encryptable functions; and
- Establish a strong negative result about the encryptability of NP-complete functions.

In particular, the authors show that the functions for which there exist efficient encryption schemes and disclose nothing are computable in expected polynomial time. For example, under certain operational assumptions, the arithmetic and logic functions can be secure, and might not disclose information concerning their operands. The authors' conclusions support the intuition that if f is hard to compute, then f is difficult to map to the range space of an encryption transform, and A cannot hide everything about f or $f(x)$ from B.

Unfortunately, the authors do not address in detail the well-known problem of disclosing the properties of encrypted vector-space operands, such as images. For example, if a function f accepts an n-pixel image **a** and returns an n-pixel image **b**, then f discloses the size n of its input, which is an important clue for the cryptanalytic adversary. Similarly, **a** or **b** may contain subimages having spatial structure that is revealed in the ciphertext. Note that this study primarily concerns customary image processing functions that compute over an n-pixel image in $\mathbf{O}(n^k)$ time, where $0 \leq k \leq 4$ is usual. Of particular implementational interest is the disclosure of salient properties of the encrypted image operands by the function f.

In summary, Abadi et al. provide a useful description of the encryptability of functions in the complexity classes NP or CoNP, which forms a conceptual basis for our work. Unfortunately, Abadi's paper contributes little to our study of the *implementational security* of polynomial-time functions (i.e., the amount of input information disclosed by a computation under practical constraints). However, a useful result is presented in the observation that polynomial-tme functions can be encrypted in polynomial time.

1.2.3.3. Key-update model of encryptive processing. In response to the shortcomings of Rivest's homomorphic functions, Ahituv, Lapid, and Neumann [44] present an incremental requirements analysis of computational security that begins with the concept of additive updates of ciphertext data with plaintext. We summarize the salient points of the analysis, as follows.

Given key $\mathbf{k}$, plaintext $\mathbf{a}$, as well as the current and updated ciphertext $\mathbf{a}_c$ and $\mathbf{a}_c^{\mathbf{new}}$, the authors discuss an update scheme given by the encryption

$$\mathbf{a}_c = (\mathbf{k} + \mathbf{a}) \bmod 2^L, \tag{8}$$

where L is a modulus that is chosen sufficiently large, in order to prevent loss of data due to overflow. Ciphertext updates with plaintext denoted by $\mathbf{a}^{\mathbf{new}}$ are given by

$$\mathbf{a}_c^{\mathbf{new}} = \mathbf{a}_c + \mathbf{a}^{\mathbf{new}}. \tag{9}$$

After j consecutive updates, the plaintext equivalent of the current ciphertext value becomes

$$\mathbf{a}_{(j)}^{\mathbf{new}} = \mathbf{a}_{c(j)}^{\mathbf{new}} - \mathbf{k}. \tag{10}$$

Unfortunately, the data is continually vulnerable to a one-time breaking of the key. Additionally, the accumulation of knowledge concerning plaintext updates could provide an adversary with increasing amounts of information about the ciphertext. Since the sum $(\mathbf{k} + \mathbf{a}_c)$ could be discovered in an intermediate state, previous updates could be traced without

discovering the key. Furthermore, the tagging of the data as plaintext or ciphertext that is required for correct decryption can provide useful information to an adversary.

The preceding situation can be improved if one employs a linear isomorphism $T: \mathbf{F}^n \times \mathbf{K} \rightarrow \mathbf{F}^n$, such that $\mathbf{a}_c = T(\mathbf{a},\mathbf{k})$, and $\mathbf{b}_c = T(\mathbf{b},\mathbf{k})$. Ahituv states the obvious requirement that

$$T^{-1}(\mathbf{a}_c + \mathbf{b}_c) = T^{-1}(T(\mathbf{a}) + T(\mathbf{b})) = T^{-1}(T(\mathbf{a} + \mathbf{b})) = \mathbf{a} + \mathbf{b}\,, \tag{11}$$

where $\mathbf{b}$ denotes plaintext to be added to $\mathbf{a}$. The utility of such encryptive addition could be augmented by multiplication if the following distributive property was supported

$$\mathbf{a}_c(\mathbf{k}) \cdot (\mathbf{a} + \mathbf{b}) = \mathbf{a}_c(\mathbf{k}) \cdot \mathbf{a} + \mathbf{a}_c(\mathbf{k}) \cdot \mathbf{b}\,, \tag{12}$$

where $\mathbf{a}_c(\mathbf{k})$ denotes ciphertext produced with key $\mathbf{k}$. Since each successive encryption would add the value of the key, it is required that after the n-th update, the most recent plaintext update is given by

$$\mathbf{a}^{new} = \mathbf{a}_c^{new} - \mathrm{n} \cdot \mathbf{k}\,. \tag{13}$$

The system described by Equations 11–13 no longer distinguishes plaintext from ciphertext in the sum, thus reducing vulnerability. However, discovery of the key can compromise the system at any time. This limitation necessitates frequent key replacement for both plaintext updates and stored ciphertext data. Although linear homomorphisms can overcome the key generation problem, such encryptions are easily broken, as discussed in Section 1.2.3.1.

Due to key vulnerability, the authors propose a further improvement whereby different neighborhoods of the keyword can be utilized as update keys. Thus, if a certain key is broken, the adversary obtains one-time (temporary) information, but cannot derive the updates $\mathbf{a}_c^{new}$ or the plaintext $\mathbf{a}$. The success of this method depends upon the generation of a sufficiently long random key, which is feasible with off-the-shelf technology [46]. For

example, let $\mathbf{k}_1$ and $\mathbf{k}_2$ be successively-generated keys. Denoting the update function with one or two keys as $U(\mathbf{k}_1)$ or $U(\mathbf{k}_1,\mathbf{k}_2)$, and encryption requiring one or two keys as $T_{\mathbf{k}_1}$ or $T_{\mathbf{k}_1,\mathbf{k}_2}$, Equations 11 and 12 become

$$\begin{aligned} T_{\mathbf{k}_1,\mathbf{k}_2}(\mathbf{a},\mathbf{b}) &= T_{\mathbf{k}_1,\mathbf{k}_2}(\mathbf{a}) + T_{\mathbf{k}_1,\mathbf{k}_2}(\mathbf{b}) = \mathbf{a}_c + \mathbf{b}_c \\ U(\mathbf{k}_1,\mathbf{k}_2)\cdot(\mathbf{a}+\mathbf{b}) &= (U(\mathbf{k}_1)\cdot\mathbf{a}) + (U(\mathbf{k}_2)\cdot\mathbf{b}), \end{aligned} \tag{14}$$

where the decryption operation T^{-1} is specified by

$$\begin{aligned} T^{-1}_{\mathbf{k}_1,\mathbf{k}_2}[T_{\mathbf{k}_1}(\mathbf{a}) + T_{\mathbf{k}_2}(\mathbf{b})] &= T^{-1}_{\mathbf{k}_1,\mathbf{k}_2}[T_{\mathbf{k}_1,\mathbf{k}_2}(\mathbf{a}+\mathbf{b})] \\ &= T^{-1}_{\mathbf{k}_1}[T_{\mathbf{k}_1}(\mathbf{a})] + T^{-1}_{\mathbf{k}_2}[T_{\mathbf{k}_2}(\mathbf{b})] = \mathbf{a}+\mathbf{b}. \end{aligned} \tag{15}$$

Assume that n-1 updates have been processed and are contained in U_{n-1}. Let the n-th plaintext update $\mathbf{a}_n$ be encrypted as U_n. The update proceeds as follows

$$\begin{aligned} U_{n+1} &= U_{n-1} + U_n \\ \mathbf{k}_{n+1} &= \mathbf{k}_{n-1} + \mathbf{k}_n, \end{aligned} \tag{16}$$

where decryption is given by

$$\mathbf{a}_{n+1} = U_{n+1} - \mathbf{k}_{n+1}. \tag{17}$$

Further security can be provided by permuting blockwise partitions of the keys to change their relative locations. In this respect, Ahituv's method suggests possible derivation from the block transpositions inherent in the DES algorithm.

Unfortunately, the previous method requires that the protocol between the client (user) and server (secure machine) become more complicated and, therefore, more awkward. Additionally, a more complex key management policy is required, which can render the system more vulnerable. Implementationally, key protection may be insufficient to prevent determination of the plaintext from knowledge of successive ciphertext updates. Thus, we progress to Yu and Yu's cipher [45], as discussed in the following section.

1.2.3.4. Yu and Yu's time-reversal ciphers.

Following Ahituv's analysis, Yu and Yu [45] proposed a method of recursive encryption T that supports the following superposition of plaintext $\mathbf{a}$ and $\mathbf{b}$ in terms of ciphertext $\mathbf{a}_c = T(\mathbf{a})$ and $\mathbf{b}_c = T(\mathbf{b})$

$$\mathbf{c}_c = (m_1 \cdot \mathbf{a}_c) + (m_2 \cdot \mathbf{b}_c), \tag{18}$$

where m_1 and m_2 are constants. The authors employ a time-reversal encryption, which is both computationally efficient and invertible, and is described as follows.

Let $\mathbf{a} \in \mathbf{F}^n$ denote a plaintext tuple to be encrypted by a time-reversal transform $T: \mathbf{F}^n \times \mathbf{K} \rightarrow \mathbf{F}^n$, such that $\mathbf{a}_c = T(\mathbf{a},\mathbf{k})$, where the key $\mathbf{k} \in \mathbf{K}$. Now, $\mathbf{a}_c(i)_{(t)}$, the i-th element of the ciphertext vector $\mathbf{a}_c$ at time t, is determined by a function of form

$$\mathbf{a}_c(i)_{(t)} = \left(\mathbf{k}\left(\mathbf{a}_c(i)_{(t)}\right) - \mathbf{a}_c(i)_{(t)}\right) \bmod q, \tag{19}$$

where q is an arbitrary modulus, and the key $\mathbf{k}(\mathbf{a}_c)$ is a function of $\mathbf{a}_c(i)$ and its L neighboring elements. Time-reversal encryption can exhibit high data security, since the cardinality of a keyspace $\mathbf{K}$ constructed from L terms about $\mathbf{a}_c(i)$ is equal to q^{q^L} elements. For example, let q equal the usual value of 256 and L=3, with a polynomially-derived key

$$\mathbf{k}(i) = c_1 \cdot \mathbf{a}_c(i-1) + c_2 \cdot \mathbf{a}_c(i) + c_3 \cdot \mathbf{a}_c(i+1), \tag{20}$$

where c_j, j=1..3, denote constants. Then $|\mathbf{K}| = 256^{256^3} \approx 2^{1.34 \times 10^8}$ possible keys.

As discussed by the authors, circulant boundary conditions should be employed, i.e., for n input data, the index i must be computed modulo n. Additionally, one must set certain initial conditions, such as

$$\mathbf{a}(i)_{(t=0)} = \mathbf{d}(i) \quad \text{and} \quad \mathbf{a}_c(i)_{(t=1)} = \mathbf{a}(i), \tag{21}$$

where elements of the arbitrary *background vector* $\mathbf{b} \in \mathbf{F}^n$ are chosen pseudorandomly.

Decryption is obtained by reversing the encryption's temporal order, i.e.,

$$\mathbf{a}_c(i)_{(t-1)} = \left(\mathbf{k}\left(\mathbf{a}_c(i)_{(t)}\right) - \mathbf{a}_c(i)_{(t+1)}\right) \bmod q. \tag{22}$$

Thus, decryption reverses the order of key addition.

Although the keyspace is large, time-reversal encryption exhibits two significant disadvantages, as follows:

1. From the definition of data superposition, T is a linear homomorphism, and can therefore be compromised by the method portrayed in Section 1.1.3.1. In particular, if L terms are used to construct a given key, the authors show that the key-generating function $\mathbf{k}$ can be determined by solving L+1 simultaneous linear equations if $\mathbf{a}_c(t-1)$ is known. However, the term $\mathbf{a}_c(t-1)$ supposedly remains confidential, since only two consecutive encrypted texts are publicly available.
2. The term $\mathbf{a}_c(t-1)$ could be plaintext, per the initial condition portrayed in Equation 22. However, deep concealment of $\mathbf{a}_c(t-1)$ is feasible a priori via extensive iteration of the encryption function, as well as appropriate selection of a background vector.

An additional method of attack on the time reversal cipher is termed *data subtraction*, which is especially attractive for decrypting linear homomorphisms. As an example of data subtraction, consider the Vernam cipher, which generates a ciphertext bit stream via the expression

$$\mathbf{a}_c = (\mathbf{k} - \mathbf{a}) \bmod 2, \tag{23}$$

where $\mathbf{a}, \mathbf{a}_c, \mathbf{k} \in \{0,1\}^n$ denote the plaintext, ciphertext, and key, respectively. Now, let us generate the updated ciphertext $\mathbf{a}_c^{new} = (\mathbf{k} - \mathbf{a}^{new}) \bmod 2$, and subtract $\mathbf{a}_c^{new}$ from $\mathbf{a}_c$ to obtain

$$\mathbf{a}_c - \mathbf{a}_c^{new} = (\mathbf{a}^{new} - \mathbf{a}) \bmod 2, \tag{24}$$

which is equivalent to the Vernam encryption of $\mathbf{a}$ using $\mathbf{a}^{new}$ as the key. If $\mathbf{a}$ and $\mathbf{a}^{new}$ are partially known, then the key can be obtained as follows

$$\mathbf{k} = (\mathbf{a}_c + \mathbf{a}) \bmod 2 . \tag{25}$$

Since the time-reversal cipher is similar to the Vernam cipher, the authors note in passing that the difference of two ciphertexts $\mathbf{a}_c$ and $\mathbf{b}_c$ is given by

$$\mathbf{a}_c(t+1) - \mathbf{b}_c(t+1) = (\mathbf{k}(\mathbf{a}_c(t)) - \mathbf{k}(\mathbf{b}_c(t))) - (\mathbf{a}_c(t-1) + \mathbf{b}_c(t-1)) . \tag{26}$$

If the key difference $\mathbf{k}(\mathbf{a}_c(t)) - \mathbf{k}(\mathbf{b}_c(t))$ vanishes (which is claimed to be infrequent) and $\mathbf{a}_c$(t-1) and $\mathbf{b}_c$(t-1) are plaintext vectors (per item 2, above), then the encryption at time t+1 could be unsuccessful. Similarly, if the sum of the ciphertexts at time t-1 vanishes, then the key difference can be revealed in the left member of Equation 27. However, the authors claim that this deficiency can be remedied by subjecting the plaintext to several recursively-applied encryption steps prior to computing the plaintext-ciphertext update. The authors further state (without proof) that there exists no simple cryptanalysis method by which one can determine either the plaintext or the keyspace from recursively-encrypted ciphertext.

As noted previously, superposition based upon time reversal is a linear homomorphism. Such transformations are notably susceptible to correlation-based attacks. That is, due to the linearity property, the ciphertext and plaintext exhibit high autocorrelation as well as high cross-correlation. Thus, successive updates can be cross-correlated to provide significant information about the keyspace and the source plaintext. Numerous reports of correlation-based cryptanalysis exist in the open literature [30], to which the reader is referred for a detailed explanation of technique.

1.2.4. Compressive Processing

Our survey of the compressive processing literature is brief, due to the recent and limited nature of work in this area. The allied (and more general) topic of processing transformed imagery is overviewed, beginning with the well-known and nearly trivial case of processing reduced matrices [10], which has been extensively researched. We then progress to Healy's work on edge detection over locally-averaged and subsampled imagery that furnished a portion of the early conceptual and analytical basis for this study. Jawerth's feature detection algorithm for wavelet-transformed imagery [47] is reviewed as is Jolion's brief discussion of operations over pyramid-encoded images [48]. We conclude our overview with a summary of Cosman et al.'s recently-reported work in pattern recognition and low-level processing over vector-quantized imagery [1].

1.2.4.1. Processing of reduced matrices. Let $\mathbf{a} \in \mathbf{R}_{m \times n}$ denote an $m \times n$-pixel, real-valued matrix. Assume that *range*($\mathbf{a}$) contains values in a set S that are deemed insignificant, i.e., can be ignored. When such values are removed from $\mathbf{a}$, we can concatenate the remaining values of $\mathbf{a}$ together with a representation of the coordinates at which such values occur, to yield a *reduced matrix* $\mathbf{a}_c$. In practice, $\mathbf{a}_c$ can be implemented as an abstract data structure (ADS), e.g., a list $\mathbf{a}_c$ whose elements are each tuples of form $(\mathbf{x}, \mathbf{a}(\mathbf{x}))$, where $\mathbf{x}$ denotes a point in *domain*($\mathbf{a}$).

Assume that we have an image operation $\bigcirc : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$, such that $\hat{\mathbf{a}} = \bigcirc(\mathbf{a})$. By processing the second coordinate of each element in $\mathbf{a}_c$ with $\bigcirc$, we can implement the pointwise operation $\bigcirc$ over the list $\mathbf{a}_c$, thereby yielding the transformed ADS $\widehat{\mathbf{a}_c} = \bigcirc(\mathbf{a}_c)$, which can be expressed as

$$\widehat{\mathbf{a}_c} = \bigcup_{\mathbf{x} \in p_1(\mathbf{a}_c)} (\mathbf{x}, \widehat{\mathbf{a}_c}(\mathbf{x})) = \bigcup_{\mathbf{x} \in p_1(\mathbf{a}_c)} (\mathbf{x}, \bigcirc (p_2[\mathbf{a}_c(\mathbf{x})])), \tag{27}$$

where p_k denotes projection onto the k-th coordinate. An identical result would be obtained if $\bigcirc$ was applied to all elements of $\mathbf{a}$ to yield $\hat{\mathbf{a}}$, and the matrix reduction was subsequently conducted by removing $\bigcirc(S)$ from $\mathbf{a}$ to yield $\hat{\mathbf{a}_c}$.

For example, let $\mathbf{a} = (0,1,2,5,3,0,0,0,3)$, and let the negligible values $S = \{0,1\}$. The list representation of the compressed image $\mathbf{a}_c$ would be given by

$$\mathbf{a}_c = ((3,2),(4,5),(5,3),(9,3)). \tag{28}$$

Let the operation $\bigcirc(f) = 2f$, where f is a value in *range*($\mathbf{a}$). Then, we have that

$$\begin{aligned} \hat{\mathbf{a}} &= \bigcirc(\mathbf{a}) = (0,2,4,10,6,0,0,0,6) \quad \text{and} \\ \hat{\mathbf{a}_c} &= ((3,\bigcirc[2]),(4,\bigcirc[5]),(5,\bigcirc[3]),(9,\bigcirc[3])) \\ &= ((3,4),(4,10),(5,6),(9,6)). \end{aligned} \tag{29}$$

Now, $\bigcirc(S) = \{\bigcirc(0), \bigcirc(1)\} = \{0,2\}$. If we remove $\bigcirc(S)$ from $\hat{\mathbf{a}}$, we obtain the list representation

$$\hat{\mathbf{a}}_c = ((3,4),(4,10),(5,6),(9,6)), \tag{30}$$

which is given in Equation 29.

Thus, the preceding example of matrix reduction can be loosely thought of as a homomorphism that preserves the operation $\bigcirc$, which was defined previously. However, the operation $\bigcirc'$ that computes over $\mathbf{a}_c$ is formulated as $\bigcirc(p_2(\mathbf{a}_c))$, per Equation 27. Thus, an operation over a reduced matrix can be derived from the corresponding operation over a sparse matrix. In Chapter 9, we show that numerous image processing operations can be computed efficiently over reduced matrices. However, matrix reduction is not an *isomorphism*, since there exists no inverse (i.e., an exact matrix expansion) that can recover the information lost when the negligible value set S contains more than one value. In Chapter 9, we describe the approximate inversion of a reduced matrix $\mathbf{a}_c$ in terms of an approximation to a source matrix $\mathbf{a}$. We call such inversion operations *approximate inverse transforms*, and provide further theoretical development in Chapters 3–5. Additionally,

we show how image operations over the reduced matrix $\mathbf{a}_c$ propagate the error due to the previously-mentioned information loss.

The chief advantage of processing reduced matrices is the decreased computational cost that accrues from the processing of fewer matrix elements. The compression ratios achievable with matrix reduction often range from 100:1 to 1000:1, which can yield computational speedups for pointwise operations that approach the compression ratio. As noted previously, the actual speedup is dependent upon the overhead of retrieving the source matrix values from the ADS that represents the reduced matrix.

As an example, consider image and matrix operations (denoted by $\bigcirc$, for purposes of brevity) such as edge detection, matrix inversion, and LU decomposition. Such operations usually require that the reduced matrix $\mathbf{a}_c$ be represented in terms of a data structure that is based upon the dataflow structure of the operation $\bigcirc$. For instance, assume that the application of $\bigcirc$ to the $\mathbf{x}$-th element of the source matrix $\mathbf{a}$ requires retrieval of the nearest neighbors of the $\mathbf{x}$-th element. It follows that the ADS that encodes the reduced matrix $\mathbf{a}_c$ should be organized to facilitate optimal retrieval of the nearest neighbors of $\mathbf{a}(\mathbf{x})$. In practice, however, when one conducts heterogeneous operations over a given ADS, certain operations must be modified slightly to yield a more efficient computation over the given ADS. In Chapter 9, we discuss the general problem of ADS derivation and the specific problem of adapting reduced-matrix operations to accept certain data structures.

We next consider the processing of images that are compressed by combination of adjacent pixel values.

1.2.4.2. Edge detection over averaged imagery. In the early 1980s, Healy [49] developed two methods for edge detection over compressed imagery. The first approach was based on an image restoration technique that optimally reversed the image compression process. The second approach was based upon a maximum-likelihood estimation of the original full-sized edge, which could be deduced from patterns in the reduced imagery. Healy showed

that, for a finite number of edge patterns over small neighborhoods, it was feasible to use a lookup table (LUT) to map the compressed edge pattern to the corresponding image neighborhood that produced the edge.

Images were reduced by mapping 4×4-pixel source blocks into 2×2-pixel blocks via local averaging, selective averaging (i.e., without averaging across edges), periodic sampling, and gradient-preserving sampling. The reduced images were edge enhanced using Kirsch or Roberts operators. Note that passive FLIR imagery (of resolution 112 lines with 120 pixel per line) was employed, which generally exhibits large, bland regions bounded by nonlinear transitions, and is frequently degraded by noise resulting from the imaging process. Thus error and noise quantification was a key study goal.

In addition to edge quality comparisons, mean-squared error (MSE) was calculated for all images, especially those produced by decompressing reduced images via pixel replication. Note that the averaging of 2×2-pixel blocks is equivalent to representing each block by its local mean, thereby guaranteeing minimum MSE with respect to image intensity. Healy states that the previously-described simple averaging method showed the best overall performance in the edge image, both quantitatively (using computed measures of image clarity) and qualitatively (via visual assessment). Healy tested several additional image reduction techniques, all of which exhibited reduced MSE. However, only simple averaging produced results that consistently corresponded to reduced MSE in the edge image.

When the Kirsch operator was applied to reconstructed (decompressed) imagery, edge detection was improved, as opposed to operating on the reduced images. For example, for the simple averaging method of image compression, MSE decreased from 1.38 to 0.75 in the edge image. However, application of the Kirsch detector to the compressed images did not fully extract the edge information. Healy's exploratory studies represent early attempts to perform and analyze image processing operations over images compressed by a well-known method such as block averaging. We further discuss the block averaging technique in

Chapter 7. A relatively new method of edge detection, which processes wavelet-compressed imagery, is described in the following section.

1.2.4.3. Local enhancement of wavelet-compressed imagery. In 1993, Jawerth, Hilton, and Huntsberger [47] reported a simple focusing technique for wavelet decompositions. With such methods, one can select neighborhoods of interest within a source image according to region and scale parameters. Additionally, it is possible to obtain variable compression rates and enhance images via wavelet-based filtering of the compressed source image.

In summary, beginning with a definition of wavelet series, Jawerth states the wavelet decomposition of an image $\mathbf{a}$ over domain $\mathbf{X}$ as

$$\mathbf{a}(\mathbf{x}) = \sum_{\mathrm{i}} \sum_{\nu} \sum_{k} \gamma_{\nu k}^{(\mathrm{i})} \psi_{\nu k}^{(\mathrm{i})}(\mathbf{x}) \,, \; \mathbf{x} \in \mathbf{X} \,, \tag{31}$$

where i denotes the order of the wavelet coefficients in γ, ψ denotes a wavelet basis function, ν represents a scale parameter that corresponds to spatial frequency, and k denotes a positional parameter. Each of the basis functions ψ belongs to one of a finite number of families $\left\{ \psi_{\nu k}^{(\mathrm{i})} \right\}_{\nu k}$, where $\psi_{\nu k}^{(\mathrm{i})}$ exists on the approximate scale $2^{-\nu}$ and is approximately located at $2^{-\nu}k$.

Now, let $\mathfrak{F} = \left\{ \mathrm{w}_{\nu k}^{(\mathrm{i})} : \mathrm{i} \in \mathbf{Z}_{\mathrm{m}} \right\}$ denote a set of *weight factors*, which are nonnegative numbers. By multiplying the coefficients in a wavelet decomposition by the appropriate weight factors, we obtain the *weighted coefficients* $\mathrm{w}_{\nu k}^{(\mathrm{i})} \gamma_{\nu k}^{(\mathrm{i})}$, with which one can emphasize the importance of certain scales and regions of the decomposed image. The authors show that under statistical constraint, one can specify regions of an image by mean-squared error criteria and employ additional error measures that relate to the smoothness of $\mathbf{a}$ over a subset $\mathcal{R}$ of Euclidean space $\mathbf{R}^{\mathrm{n}}$. Thus, by choosing weight factors that are relatively large at certain scales and locations, one can obtain better approximations to image details in those regions.

Image compression can be effected by analyzing the weighted coefficients $w_{\nu k}^{(i)} \gamma_{\nu k}^{(i)}$ of an image, rather than the image itself. As a result, bland areas of an image, which will be manifested as coefficients at large values of the spatial frequency parameter ν, can be subtracted from the wavelet series, parameterized, and concatenated with the compressed image. Such methods are reminiscent of transform coding using sparse matrix reduction, which selects coefficients by their magnitudes. However, in wavelet-based image manipulation, one stores in the compressed image those weighted coefficients whose *spatial frequencies* are in the set of frequencies exhibited by key objects in the scene. Image enhancement is similarly performed, by increasing the magnitude of the wavelet coefficients whose spatial frequencies correspond to features of interest in the scene. A related hierarchical method, processing of pyramidal imagery, is next overviewed.

1.2.4.4. Processing of pyramidally compressed imagery. In the late 1970s, Tanimoto proposed that pyramidal data structures be employed for object detection in natural scenes [50]. We herein overview methods by which Tanimoto's theory of operations over pyramidal data structures has been adapted to operations such as edge detection.

Let $\mathbf{a}$ denote an $n \times n$-pixel source image whose pixel values comprise the set F, where n is a power of some radix r. Additionally, let *domain*($\mathbf{a}$) be tesselated into adjacent blocks of size $r \times r$ pixels. Assuming that we can reduce each block (via a data compression transform T) to $k < r^2$ values in F, a compression ratio of $CR = r^2/k$ will be obtained for each block and, therefore, for $\mathbf{a}$.

Without loss of specificity, assume that each k-pixel compressed block is represented as a terminal node of an r-ary tree. From our initial assumption that $n = r^m$, we have $\log_r(n) = m$, which implies an m-level r-ary tree representation of $\mathbf{a}$ called an *image pyramid*. In the uncompressed image $\mathbf{a}$, the terminal nodes are $r \times r$-pixel blocks. Thus, in the compressed image $\mathbf{a}_c$, the terminal nodes are k-pixel compressed versions of the terminal nodes of $\mathbf{a}$. Further assume that T combines a local feature-detection function with the block reduction

operation. Thus, a leaf of the tree that corresponds to the pyramid data structure would contain a compressed block of the source image, with edge information encoded in the compressed image or concatenated to the compressed block. Given such a structure, one can begin processing (for example, via edge detection) at the leaves. By traversing the tree in bottom-up fashion, the result (in this example, an edge map) is propagated to the root node. Thus, an edge detection operation over the pyramidal tree would process the entire tree in $\mathbf{O}(n \cdot \log_r(n))$ work. With n^2-fold parallelism, $\mathbf{O}(\log_r n)$ time would be required.

Additionally, operations such as image summation and maximum can be computed approximately over image pyramids in time proportional to the number of nodes at a given level [50]. With n^2-fold parallelism, pointwise operations would incur a speedup that is proportional to the number of pixels in the pyramid level at which processing occurs. In contrast, noncompressive processing requires $\mathbf{O}(|domain(\mathbf{a})|)$ operaions. In Chapter 3, we will see that this result is key to achieving computational efficiencies at compressive operation level.

1.2.4.5. Low-level operations on vector-quantized imagery. Let an $\mathbf{F}$-valued source image $\mathbf{a}$ on domain $\mathbf{X}$ be denoted as $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ (per the discussion of Section 2.1), and assume that $\mathbf{a}$ is segmented into k-pixel encoding blocks, also called *vectors*. A vector quantization transform $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}} \times (\mathbf{F}^{\mathbf{k}})^{\mathbf{G}}$ groups the vectors into classes that comprise a *codebook* of form $(\mathbf{F}^{\mathbf{k}})^{\mathbf{G}}$, based on a representational error criterion. The error measure determines: (a) how closely the exemplar vectors or patterns in each class resemble each other, and (b) how well the patterns represent the partitions of the source image from which each pattern class was derived. Thus, the encoded image is merely a list (or array) of the indices in $\mathbf{G}$ of codebook exemplars.

At a high level, one can correctly assert that the VQ-compressed image is the result of applying a substitution to the source image. That is, given a codebook, the VQ transform substitutes for a given encoding block the index of the pattern class that best portrays the

encoding block's spatial pattern under constraint of a prior error criterion. For example, if the image domain $\mathbf{X}$ is two-dimensional, and k-pixel encoding blocks are employed, then the VQ codebook will have no more than $\lceil |\mathbf{X}|/k \rceil$ pattern classes. Likewise, the source image will be compressed into $\lceil |\mathbf{X}|/k \rceil$ pattern indices, regardless of the number of pattern classes. If one constrains the VQ process such that the maximum number of bits required to encode each pattern index is less than or equal to the number of bits required to encode each image value, then the compression ratio $CR \geq k$.

Since the VQ transform can be thought of (at a high level) as a substitution, the processing of VQ imagery is, in principle, an easy matter. For example, an approximation to the sum of a source image may be obtained by summing the exemplar vector sums weighted by their frequency of occurrence in the compressed image. The substitutional model of VQ underlies much of the theory for processing VQ-encoded imagery, and has thus been employed in recent reports of VQ-based arithmetic [51], pattern recognition over VQ imagery [1] and, most recently, VQ-based template operations [53]. Cosman et al. have summarized their work in processing VQ-encoded imagery in Reference 1, which includes a variety of ingenious methods for using substitution-based computing of the image histogram, image enhancement by histogram equalization, an approximation to edge detection, and several methods of graphics rendering. In each case, Cosman shows that implementationally significant computational efficiencies can be obtained.

For example, let source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, a VQ codebook $\mathbf{c} \in \left(\mathbf{F}^{\mathbf{k}}\right)^{\mathbf{G}}$, and the source image histogram $\mathbf{h} \in \mathbf{N}^{\mathbf{F}}$. One can approximate $\mathbf{h}$ by computing the histogram $\overline{\mathbf{h}}_i$, $i \in G$ of each codebook vector, then weighting $\overline{\mathbf{h}}_i$ by its frequency of occurrence f_i in the VQ-compressed image. The histogram is approximated as $\mathbf{h} \approx \sum_{i \in G} f_i \cdot \overline{\mathbf{h}}_i$. In Chapter 9, we discuss the processing of VQ imagery in detail, with emphasis upon the more difficult case of VQ-based image-template operations.

1.3. Technical Approach

In this dissertation, we summarize our previous work in the classification of image transforms [2], and discuss methods for deriving analogues that implement selected image processing operations over the range spaces of such transforms. We provide specific examples that show techniques by which theoretical speedups in image and signal processing algorithms can be achieved. For example, assume that N compressed data are processed instead of M source data, where $N < M$ and M/N approximates the compression ratio (CR). In certain cases, we claim that a computational speedup of order CR is possible. In support of such claims, we emphasize

1. Description of a transform taxonomy (classification scheme) that includes multiple subclasses for the detailed characterization of commonly-employed compressive transforms.
2. Elucidation of methods for deriving analogues of common image processing operations over range spaces of multiple transforms in each taxonomic class.
3. Application of the methods obtained in Step 2 to the derivation of analogues of common image processing operations over the range spaces of two or three transforms in each class. In particular, we consider

   - a generic type of JPEG transform,
   - VQ with fixed blocksize, and
   - VPIC with a small exemplar set.

   Such transforms are shown to be practical for a variety of image compression tasks. Additionally, VPIC followed by Huffman coding is shown to perform well for fast compression applications at high compression ratios.
4. Analysis of the operations derived in Step 3 in terms of noise sensitivity, sequential speedup, and data security.

5. Illustration of various stages of the image compression and compressive computation process with images derived from practice, together with supporting performance data.
6. Feasibility analysis of compressive computational systems based upon error accumulation and computational efficiency.
7. Discussion of possible additional advantages or problems that may result from compressive or encryptive processing, which could be addressed in future research. For example, we discuss implementational issues inherent in the mapping of various compressive operations to a SIMD-parallel mesh architecture.

As a result of this study, we have achieved several advances in image processing technology that are described in the following section.

1.4. Novel Claims

We claim the following novel accomplishments:

- High-level theoretical unification of compressive and encryptive processing;
- Classification of image transformations in a rigorous, concise taxonomy, which is instrumental in facilitating the efficient design of compressive and encryptive processing algorithms;
- Computational speedup for many image processing operations that is achieved by computing with compressed data;
- Reduction in the degree of parallelism that may be possible for certain compressive operations on specific parallel architectures, due to the presence of fewer data; and
- Enhancement of data security, which may be possible by signal and image processing of data that have been compressed, then encrypted.

An implication of this dissertation is that the theory presented herein may furnish the basis for novel, fundamental advances in computational efficiency and data security not hitherto realized via existing image processing algorithms. Specific applications currently exist in areas of high-bandwidth (i.e., high frame rate) image processing, such as real-time automatic target recognition using high-speed imaging devices and real-time medical image processing in support of bone or tissue visualization for machine-assisted surgery. Additional applications include efficient search over compressed databases, as well as parallel algorithm designs that permit reduced parallelism but preserve the functionality of the corresponding image operation(s).

1.5. Implementational Advantages and Disadvantages

An interesting observation resulted from our early research in the derivation of an algorithm for performing connected component labelling on one-dimensional, n-pixel Boolean imagery compressed by runlength encoding [5]. We noted that the customary tree structure of connected component labelling [6] was transformed to a SIMD-parallel structure in certain stages of the compressive algorithm. However, an $\mathbf{O}(n)$ communications cost remained. We conjecture that similar effects may occur with certain compressive formats that encode image regions in terms of features which are key to a prespecified, tree-structured image processing algorithm.

For example, let a transform T encode an image region based upon edge or boundary information. It would be reasonable to assume that an analogue over *range*(T) of a boundary detection algorithm could be derived that would be amenable to SIMD-parallel implementation, due to preservation of boundary information in the compressed image. Thus, it is reasonable to assume that a tree-structured boundary detection algorithm could be rendered SIMD-parallel via such methods of compressive algorithm design, since boundary information is assumed to be localized in each pixel of the compressed image.

In particular, one could design or select transforms that compress the source image so as to preserve features computed by such image operations (e.g., boundary detection). If such was the case, then it could be possible to simplify the design of parallel algorithms for certain image processing operations. Since parallel algorithm design is nontrivial but parallel computation has great promise for practical applications, such developments could possibly increase the speed and scope of parallel computer applications in image processing.

There are two disadvantages of compressive processing that pertain primarily to the use of a cascaded sequence compressive operations, denoted by S. First, error due to lossy transformation of a source image **a** can accumulate as compressive computation proceeds through S(**a**), thus rendering the visual quality of the decompressed result unacceptable. Additionally, this is a problem for automated target recognition conducted over decompressed images that were previously subject to compressive processing. Second, all operations in S may not be computable over a given compressive format with efficiencies that meet the design criteria for a given implementation. Thus, conversion between compressive formats may be required, which increases overhead and decreases resultant efficiency. We address the preceding issues throughout this dissertation, with particular emphasis on computational error in the applications portion.

CHAPTER 2
REVIEW OF NOTATION

Prior to developing supporting theory, we summarize study notation. In order to acquaint the reader with the majority of our notation, an overview of the image algebra subset employed in this study is presented in Section 2.1. In Section 2.2, we discuss general rules for, and deviations from, the list of symbols contained in the front portion of this document.

2.1. Overview of the Image Algebra (IA) Subset

Image algebra is a set-theoretical and functional notation that unifies linear and nonlinear mathematics in the image domain. The basic entities of IA are set elements, sets, and functional mappings. Sets and set operations are discussed in Section 2.1.1, while functions are subdivided into several classes, namely, images (Section 2.1.2), operations on images (Section 2.1.3), nonrecursive image-template functions (Section 2.1.4), and nonrecursive operations on templates (Section 2.1.5). Recursive operations are summarized in Section 2.1.6.

2.1.1. Sets and operations upon sets

In image algebra, *set elements* are generally defined implementationally as having any data type, such as integer, real, or complex. Additionally, set elements may be data structures such as sets, lists, or images, but such unusual structures tend to require processing operations that are not well-defined and thus are not used extensively in

computational practice. Elements are collected in *sets*, which may be partially or totally ordered.

Value sets are denoted by open-face capital letters. For example, the customary value sets $\mathbb{Z}$, $\mathbb{R}$, and $\mathbb{C}$ denote the integer, real, and complex numbers, respectively. Additionally, $\mathbb{N} \equiv \mathbb{Z}^+$ denotes the positive integers and $\mathbb{F}$ denotes a generalized value set.

Point sets are denoted by bold upper-case letters from the tail of the alphabet (e.g., $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$), and are customarily subsets of Euclidean n-space $\mathbb{R}^n$.

Image algebra supports the customary *set operations* of union ($\cup$), intersection ($\cap$), set subtraction ($\setminus$), cardinality ($|\mathbb{F}|$), and choice. Set operations obey the relevant laws of identity, idempotence, associativity, distributivity, deMorgan's Law, etc.

2.1.2. Images

An *image* is a mapping of a point set (functional domain) to a value set (functional range). Images are usually denoted by bold, lower-case letters from the head of the alphabet, such as **a**, **b**, or **c**, as well as by emboldened strings, such as **ci** or **ldf**.

Given a point set $\mathbf{X}$ and a value set $\mathbb{F}$, the image $\mathbf{a} \in \mathbb{F}^{\mathbf{X}}$, called an $\mathbb{F}$-*valued image on* $\mathbf{X}$, is the function $\mathbf{a}\colon \mathbf{X} \to \mathbb{F}$. The functional domain of **a**, denoted by *domain*(**a**), is equal to $\mathbf{X}$, while *range*(**a**) = $\mathbb{F}$ denotes the functional range of **a**.

The *graph of an image* $\mathbf{a} \in \mathbb{F}^{\mathbf{X}}$, denoted by $G(\mathbf{a})$, is expressed as $\mathbf{a} \equiv G(\mathbf{a}) = \{(\mathbf{x}, \mathbf{a}(\mathbf{x}))\colon \mathbf{x} \in \mathbf{X}, \mathbf{a}(\mathbf{x}) \in \mathbb{F}\}$. Additionally, the *point* $\mathbf{x} \in \mathbf{X}$, the *value* $\mathbf{a}(\mathbf{x}) \in \mathbb{F}$, and the *pixel* $(\mathbf{x},\mathbf{a}(\mathbf{x}))$ are important features of the image **a**. The more elementary image operations are defined as follows.

2.1.3. Operations upon images

Operations upon $\mathbb{F}$-valued images are the naturally-induced operations of the algebraic system $\mathbb{F}$. For example, operations on real valued images $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{\mathbf{X}}$ are the elementary

operations induced by the vector lattice (a vector space which is a lattice) $\mathbf{R}$. Thus, basic operations on $\mathbf{R}^{\mathbf{X}}$ reflect the corresponding arithmetic and logical operations on $\mathbf{R}$.

The *generalized unary operation* $f: \mathbf{F} \to \mathbf{F}$ thus induces the *generalized unary image operation* $f: \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$, which is applied to the image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ to yield

$$\mathbf{b} = f(\mathbf{a}) \equiv \{(\mathbf{x}, \mathbf{b}(\mathbf{x})) : \mathbf{b}(\mathbf{x}) = f(\mathbf{a}(\mathbf{x})), \mathbf{x} \in \mathbf{X}\}. \tag{32}$$

For example if $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, then $\sin(\mathbf{a}) \equiv \{(\mathbf{x}, \mathbf{b}(\mathbf{x})) : \mathbf{b}(\mathbf{x}) = \sin(\mathbf{a}(\mathbf{x})), \mathbf{x} \in \mathbf{X}\}$.

Additional unary operations upon the image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ are the *image sum*, denoted by $\Sigma\mathbf{a} = \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x})$, and the *image maximum* $\vee\mathbf{a} = \bigvee_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x})$. Given an image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and an associative, commutative operation $\gamma: \mathbf{F} \times \mathbf{F} \to \mathbf{F}$, the preceding operations can be generalized in terms of the *global reduce operation* $\Gamma\mathbf{a} \equiv \prod_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x}) \equiv \mathbf{a}(\mathbf{x}_1) \gamma \mathbf{a}(\mathbf{x}_2) \gamma \cdots \gamma \mathbf{a}(\mathbf{x}_n)$, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbf{X}$.

Given images $\mathbf{a}, \mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ and a binary operation $\bigcirc: \mathbf{F} \times \mathbf{F} \to \mathbf{F}$, the *binary image operation* of Hadamard (pointwise) arithmetic $\bigcirc: \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ is defined as $\mathbf{a} \bigcirc \mathbf{b} \equiv \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) \bigcirc \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}$. Given the images where $\mathbf{a}, \mathbf{b} \in \mathbf{R}^{\mathbf{X}}$, the usual arithmetic operations $(+, -, \cdot, /)$ as well as the Boolean logic operations are supported, as shown in the following examples

$$\begin{aligned} \mathbf{a}+\mathbf{b} &\equiv \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) + \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \\ \mathbf{a}*\mathbf{b} &\equiv \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) \cdot \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \\ \mathbf{a} \vee \mathbf{b} &\equiv \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) \vee \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \end{aligned} \tag{33}$$

where $\vee$ denotes maximum. Since complex numbers lack a natural lattice structure, only addition and multiplication are defined for complex-valued images. The remaining operations on $\mathbf{R}^{\mathbf{X}}$ can be expressed in terms of the arithmetic functions, or are induced on $\mathbf{R}^{\mathbf{X}}$ by the corresponding operations on $\mathbf{R}$. For example, exponentiation of the previously-defined images $\mathbf{a}$ and $\mathbf{b}$ is given by

$$\mathbf{a}^{\mathbf{b}} \equiv \left\{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x})^{\mathbf{b}(\mathbf{x})} \text{ if } \mathbf{a}(\mathbf{x}) \neq 0, \text{ otherwise } \mathbf{c}(\mathbf{x}) = 0, \mathbf{x} \in \mathbf{X}\right\}. \tag{34}$$

Note that exponentiation is defined when $\mathbf{a(x)} = 0$. Computation of the logarithm, which is invalid for zero and the negative real values, is given by

$$\log_{\mathbf{a}}(\mathbf{b}) \equiv \{(\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \log_{\mathbf{a}(\mathbf{x})}(\mathbf{b}(\mathbf{x})) \text{ if } \mathbf{a}(\mathbf{x}),\mathbf{b}(\mathbf{x}) > 0\,,\ \mathbf{x} \in \mathbf{X}\}\,. \tag{35}$$

Given the *constant image* $\mathbf{a} = \{(\mathbf{x}, \mathbf{a}(\mathbf{x})) : \mathbf{a}(\mathbf{x}) = k,\ k \in \mathbf{F},\ \mathbf{x} \in \mathbf{X}\}$, we have that

$$\begin{aligned} &\mathbf{b}^k \equiv \mathbf{b}^{\mathbf{a}} \text{ and } k^{\mathbf{b}} \equiv \mathbf{a}^{\mathbf{b}} \\ &k\mathbf{b} \equiv \mathbf{a} * \mathbf{b} \text{ and } k + \mathbf{b} \equiv \mathbf{a} + \mathbf{b}\,. \end{aligned} \tag{36}$$

Constant images which are of special importance in IA include the zero image $\mathbf{0} \equiv \{(\mathbf{x}, 0) : \mathbf{x} \in \mathbf{X}\}$ and the unitary image $\mathbf{1} \equiv \{(\mathbf{x}, 1) : \mathbf{x} \in \mathbf{X}\}$. For example, notational brevity is realized by defining the image sum as $\Sigma\mathbf{a} = \sum_{\mathbf{x}\in\mathbf{X}} \mathbf{a}(\mathbf{x}) = \mathbf{a} \bullet \mathbf{1}$, where the dot product $\mathbf{a} \bullet \mathbf{b} = \sum_{\mathbf{x}\in\mathbf{X}} \mathbf{a}(\mathbf{x}) \cdot \mathbf{b}(\mathbf{x})$.

Subtraction, division, and minimum are defined in terms of the basic operations and the constant image -1, where $-\mathbf{b} = -1 * \mathbf{b}$, and

$$\mathbf{a} - \mathbf{b} = \mathbf{a} + (-\mathbf{b}),\ \mathbf{a}/\mathbf{b} = \mathbf{a} * \mathbf{b}^{-1}, \text{ and } \mathbf{a} \wedge \mathbf{b} = -(-\mathbf{a} \vee -\mathbf{b})\,. \tag{37}$$

The images $\mathbf{0}$ and $\mathbf{1}$ exhibit the identity properties $\mathbf{a}+\mathbf{0} = \mathbf{a}$ and $\mathbf{a}*\mathbf{1} = \mathbf{a}$. Although $\mathbf{b}*\mathbf{b}^{-1}*\mathbf{b} = \mathbf{b}$, we note that $\mathbf{b}*\mathbf{b}^{-1}$ is not necessarily equal to $\mathbf{1}$. Although this usage is infrequent, we occasionally call $\mathbf{b}^{-1}$ the *pseudo-inverse* of $\mathbf{b}$, as discussed in Reference 54.

Given image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and $S : \mathbf{X} \to 2^{\mathbf{X}}$, the *generalized characteristic function*

$$\chi_S(\mathbf{a}) = \left\{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{a}(\mathbf{x}) \in S(\mathbf{x}) \\ 0 \text{ otherwise} \end{cases},\ \mathbf{x} \in \mathbf{X}\right\} \tag{38}$$

can be employed in thresholding $\mathbf{a}$ at a value $T \in \mathbf{F}$, as follows

$$\chi_{>T}(\mathbf{a}) = \left\{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{a}(\mathbf{x}) > T \\ 0 \text{ otherwise} \end{cases},\ \mathbf{x} \in \mathbf{X}\right\}. \tag{39}$$

Characteristic functions may also be defined in terms of elementary IA operations, as shown in Reference 54.

The *restriction* of image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ to a subset $\mathbf{W}$ of $\mathbf{X}$ is denoted by $\mathbf{a}|_{\mathbf{W}} \equiv \{(\mathbf{x},\mathbf{a}(\mathbf{x}))\colon \mathbf{x} \in \mathbf{W}\}$. The *extension* of image $\mathbf{b} \in \mathbf{F}^{\mathbf{W}}$ to the image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, where $\mathbf{W} \subset \mathbf{X}$, is given by

$$\mathbf{b}|^{\mathbf{a}} = \left\{(\mathbf{x},\mathbf{c}(\mathbf{x}))\colon \mathbf{c}(\mathbf{x}) = \begin{cases} \mathbf{a}(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbf{X}\backslash\mathbf{W} \\ \mathbf{b}(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbf{W} \end{cases}\right\}. \tag{40}$$

Implementationally, restriction and extension can be thought of as *cut* and *paste* operations, respectively.

Furthermore, a restriction of $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ to a subset of $\mathbf{a}$ whose values are characterized by a property S of $\mathbf{F}$ is given by

$$\mathbf{a}\|_{\mathbf{S}} \equiv \{(\mathbf{x},\mathbf{a}(\mathbf{x}))\colon \mathbf{a}(\mathbf{x}) \in \mathbf{S},\ \mathbf{x} \in \mathbf{X}\}. \tag{41}$$

For example, given a threshold $\mathbf{T} \in \mathbf{R}$, if $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, then $\mathbf{a}\|_{>\mathbf{T}} \equiv \{(\mathbf{x},\mathbf{a}(\mathbf{x}))\colon \mathbf{a}(\mathbf{x}) > \mathbf{T},\ \mathbf{x} \in \mathbf{X}\}$.

2.1.4. Nonrecursive templates and image-template operations

Templates are powerful constructs that map a point to an image. Templates unify the customary concepts of convolution masks, sampling windows, structuring elements (mathematical morphology), and neighborhood functions.

For example, let $\mathbf{X}$ and $\mathbf{Y}$ denote point sets, P denote a set of parameters, and $\mathbf{F}$ denote the value set. A *generalized* $\mathbf{F}$*-valued template* $\mathbf{t}$ *from* $\mathbf{Y}$ *to* $\mathbf{X}$ *with parameters in* P is a function of form $\mathbf{t}\colon \mathrm{P} \rightarrow (\mathrm{F}^{\mathbf{X}})^{\mathbf{Y}}$. For purposes of notational simplicity, let $\mathbf{t}(\mathrm{p})$ denote a function from $\mathbf{Y}$ to $\mathbf{F}^{\mathbf{X}}$, for each $\mathrm{p} \in \mathrm{P}$. The template *values*, of form $\mathbf{t}_{\mathbf{y}}(\mathrm{p}) \in \mathbf{F}^{\mathbf{X}}$, where the *target point* $\mathbf{y} \in \mathbf{X}$, are each an $\mathbf{F}$-valued image on $\mathbf{X}$. If the set P of parameters is unspecified or is implicit in the context of a discussion, then $\mathbf{t}_{\mathbf{y}}$ denotes $\mathbf{t}(\mathrm{p})(\mathbf{y})$, and $(\mathrm{F}^{\mathbf{X}})^{\mathbf{Y}}$ denotes $\mathrm{P} \rightarrow (\mathrm{F}^{\mathbf{X}})^{\mathbf{Y}}$. Given $\mathbf{t}_{\mathbf{y}} \in \mathbf{F}^{\mathbf{X}}$, the *weights* of $\mathbf{t}$ are denoted by $\mathbf{t}_{\mathbf{y}}(\mathbf{x}) \in \mathbf{F}$, $\mathbf{x} \in \mathbf{X}$.

If $\mathbf{t}$ is a real-valued template from $\mathbf{Y}$ to $\mathbf{X}$, then the *support of* $\mathbf{t_y}$ is defined as $S(\mathbf{t_y}) = \{\mathbf{x} \in \mathbf{X}\colon \mathbf{t_y}(\mathbf{x}) \neq 0\}$. The set $S(\mathbf{t_y}) = \{\mathbf{x} \in \mathbf{X}\colon \mathbf{t_y}(\mathbf{x}) \neq -\infty\}$ is referred to as the *configuration of* $\mathbf{t}$ *at the point* $\mathbf{y}$.

If $\mathbf{X}$ is a space with operation $+$, then a template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ is called *translation-invariant* (with respect to the operation $+$) if and only if for each triple $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{X}$, that $\mathbf{t_y}(\mathbf{x}) = \mathbf{t_{y+z}}(\mathbf{x} + \mathbf{z})$. If a template is *translation-variant*, then it is not translation-invariant. For purposes of brevity we call translation-invariant templates *invariant templates*, and translation-variant templates are called *variant templates*.

Given an image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and a template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$, as well as the associative, commutative functions $\gamma, \circ\colon \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$, the *backward image-template operation*, also called the *right product*, is given by

$$\mathbf{b} = \mathbf{a} \textcircled{\gamma} \mathbf{t} \equiv \left\{(\mathbf{y}, \mathbf{b}(\mathbf{y}))\colon \mathbf{b}(\mathbf{y}) = \bigGamma_{\mathbf{x} \in \mathbf{X}} (\mathbf{a}(\mathbf{x}) \circ \mathbf{t_y}(\mathbf{x})),\ \mathbf{y} \in \mathbf{Y}\right\}. \tag{42}$$

When $\mathbf{F} \subset \mathbf{C}$, as is customary, the right product can be instantiated as the image-template convolution

$$\mathbf{b} = \mathbf{a} \oplus \mathbf{t} \equiv \left\{(\mathbf{y}, \mathbf{b}(\mathbf{y}))\colon \mathbf{b}(\mathbf{y}) = \sum_{\mathbf{x} \in \mathbf{X}} (\mathbf{a}(\mathbf{x}) \cdot \mathbf{t_y}(\mathbf{x})),\ \mathbf{y} \in \mathbf{Y}\right\}, \tag{43}$$

by setting $\gamma = +$ and $\circ = \cdot$. For $\mathbf{t} \in (\mathbf{F}^{\mathbf{Y}})^{\mathbf{X}}$, the *forward image-template operation* or *left product*, denoted by $\mathbf{t} \oplus \mathbf{a}$, is given by

$$\mathbf{t} \oplus \mathbf{a} \equiv \left\{(\mathbf{y}, \mathbf{c}(\mathbf{y}))\colon \mathbf{c}(\mathbf{y}) = \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{t_x}(\mathbf{y}) \cdot \mathbf{a}(\mathbf{x}),\ \mathbf{y} \in \mathbf{Y}\right\}. \tag{44}$$

In either case, $\mathbf{b} \in \mathbf{F}^{\mathbf{Y}}$. Ritter [55] showed that computation of the generalized convolution can be rendered more efficient by replacing $\mathbf{X}$ with $S(\mathbf{t_y})$ in Equation 43, or with $S(\mathbf{t_x})$ in Equation 45. For example, in Equation 45, $\sum_{\mathbf{x} \in S(\mathbf{t_x})} \mathbf{t_x}(\mathbf{y}) \cdot \mathbf{a}(\mathbf{x}) = 0$ whenever $S(\mathbf{t_x}) = \varnothing$.

The three elementary image-template operations employed in the transformation of real-valued images are called generalized convolution ($\oplus$), multiplicative maximum ($\bigvee$, where $\gamma = \vee$ and $\circ = \cdot$), and additive maximum ($\boxvee$, where $\gamma = \vee$ and $\circ = +$). For example,

if $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$ are defined in terms of the value sets known as the *extended real numbers* ($\mathbf{R}_{-\infty} = \mathbf{R} \cup \{-\infty\}$ and $\mathbf{R}_{+\infty} = \mathbf{R} \cup \{+\infty\}$), then we have the following nonlinear operations

$$\mathbf{b} = \mathbf{a} \textcircled{\wedge} \mathbf{t} \equiv \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \colon \mathbf{b}(\mathbf{y}) = \bigvee_{\mathbf{x} \in S(\mathbf{t}_\mathbf{y})} (\mathbf{a}(\mathbf{x}) \cdot \mathbf{t}_\mathbf{y}(\mathbf{x})),\ \mathbf{y} \in \mathbf{Y} \right\}, \tag{45}$$

and

$$\mathbf{b} = \mathbf{a} \boxtimes \mathbf{t} \equiv \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \colon \mathbf{b}(\mathbf{y}) = \bigvee_{\mathbf{x} \in S(\mathbf{t}_\mathbf{y})} (\mathbf{a}(\mathbf{x}) + \mathbf{t}_\mathbf{y}(\mathbf{x})),\ \mathbf{y} \in \mathbf{Y} \right\}. \tag{46}$$

The functions of multiplicative and additive minimum ($\textcircled{\wedge}$ and $\boxtimes$, respectively) are defined symmetrically to Equations 46 and 47. The operations $\textcircled{\vee}$ and $\boxtimes$, together with a larger class of generalized image-template operators, are discussed in detail in Reference 54.

Templates can implement mappings across heterogeneous point sets, and are thereby useful for coordinate transformations, such as image rotation, magnification, or changes in dimensionality. In the latter case, applications to multispectral imagery and sensor fusion have been extensive [56].

Key to the utilization of the nonlinear functions $\textcircled{\vee}$ and $\boxtimes$ in applications of greater complexity is the concept of *conjugacy*, which is described as follows. Recall the preceding definitions of IA operations in terms of basic operations, as well as our previous mention of the pseudoinverse. Likewise, recall that operations on $\mathbf{F}^{\mathbf{X}}$ are induced by the corresponding operations on $\mathbf{F}$. Thus, one can see that the ring $(\mathbf{R}^{\mathbf{X}}, +, *)$ and the lattice $(\mathbf{R}^{\mathbf{X}}, \vee)$ behave much like the ring and lattice of real numbers. Similar observations hold for the extended real-valued images. Given the notion of an additive or multiplicative inverse, the concepts of additive or multiplicative conjugacy are suggested.

For example, if $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}_{-\infty}$ or $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}_{+\infty}$, and we assume that $-(+\infty) = -\infty$ and $-(-\infty) = +\infty$, then the *additive conjugate of* $\mathbf{a}$, denoted by $\mathbf{a}^*$, is defined as $\mathbf{a}^*(\mathbf{x}) = -(\mathbf{a}(\mathbf{x}))$, $\mathbf{x} \in \mathbf{X}$. Thus, $(\mathbf{a}^*)^* = \mathbf{a}$, and if $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}_{-\infty}$, then $\mathbf{a}^* \in \mathbf{R}^{\mathbf{X}}_{+\infty}$, which is the conjugate space.

Denote $\mathbf{R}^{\geq 0}_{+\infty} = \mathbf{R}^{+} \cup \{0, +\infty\}$. For $\mathbf{a} \in \left(\mathbf{R}^{\geq 0}_{+\infty}\right)^{\mathbf{X}}$, the *multiplicative conjugate of* $\mathbf{a}$ is denoted by $\overline{\mathbf{a}}$ and is defined as

$$\overline{\mathbf{a}}(\mathbf{x}) = \begin{cases} 1/\mathbf{a}(\mathbf{x}) & \text{if } \mathbf{a}(\mathbf{x}) \neq 0 \text{ or } \mathbf{a}(\mathbf{x}) \neq +\infty \\ +\infty & \text{if } \mathbf{a}(\mathbf{x}) = 0 \\ 0 & \text{if } \mathbf{a}(\mathbf{x}) = +\infty \end{cases}, \mathbf{x} \in \mathbf{X}. \tag{47}$$

Likewise, several template inversions are key to the expression of image transformations. The *transpose of a template* $\mathbf{t} \in \left(\mathbf{F}^{\mathbf{X}}\right)^{\mathbf{Y}}$ is the template $\mathbf{t}' \in \left(\mathbf{F}^{\mathbf{Y}}\right)^{\mathbf{X}}$, which is defined in terms of its weights as $\mathbf{t}'_{\mathbf{x}}(\mathbf{y}) = \mathbf{t}_{\mathbf{y}}(\mathbf{x})$, $\mathbf{x} \in \mathbf{X}$, $\mathbf{y} \in \mathbf{Y}$. The following conjugate templates are defined for the extended real valued numbers. If $\mathbf{t} \in \left(\mathbf{R}^{\mathbf{X}}_{-\infty}\right)^{\mathbf{Y}}$ (or $\mathbf{t} \in \left(\mathbf{R}^{\mathbf{X}}_{+\infty}\right)^{\mathbf{Y}}$), then the *additive conjugate of* $\mathbf{t}$ is the template $\mathbf{t}^{*} \in \left(\mathbf{R}^{\mathbf{Y}}_{+\infty}\right)^{\mathbf{X}}$ (or $\mathbf{t}^{*} \in \left(\mathbf{R}^{\mathbf{Y}}_{-\infty}\right)^{\mathbf{X}}$), such that $\mathbf{t}^{*}_{\mathbf{x}}(\mathbf{y}) = -\mathbf{t}'_{\mathbf{y}}(\mathbf{x})$, $\mathbf{x} \in \mathbf{X}$, $\mathbf{y} \in \mathbf{Y}$. If $\mathbf{t} \in \left(\mathbf{R}^{\mathbf{X}}_{+\infty}\right)^{\mathbf{Y}}$, then the *multiplicative conjugate of* $\mathbf{t}$ is the template $\overline{\mathbf{t}} \in \left(\mathbf{R}^{\mathbf{Y}}_{+\infty}\right)^{\mathbf{X}}$, such that $\overline{\mathbf{t}}_{\mathbf{x}}(\mathbf{y}) = \overline{[\mathbf{t}_{\mathbf{y}}(\mathbf{x})]}$, $\mathbf{x} \in \mathbf{X}$, $\mathbf{y} \in \mathbf{Y}$, after Equation 48.

With template inversion, the operations $\bigcirc\!\!\!\!\vee$ and $\boxtimes$ are employed in the definition of the conjugate operations in a manner that is reminiscent of DeMorgan's Law, i.e.,

$$\mathbf{a} \boxtimes \mathbf{t} \equiv (\mathbf{t}^{*} \boxtimes \mathbf{a}^{*})^{*} \tag{48}$$

and

$$\mathbf{a} \bigcirc\!\!\!\!\wedge \mathbf{t} = \overline{(\overline{\mathbf{t}} \bigcirc\!\!\!\!\vee \overline{\mathbf{a}})}. \tag{49}$$

2.1.5. Nonrecursive operations on templates

The elementary arithmetic operations on images given in Equation 34 also generalize to templates. If $\mathbf{s}$ and $\mathbf{t}$ are real-valued templates from $\mathbf{Y}$ to $\mathbf{X}$, then pointwise template arithmetic is defined as follows

$$\begin{aligned} \mathbf{s}+\mathbf{t} &\text{ is defined by } (\mathbf{s}+\mathbf{t})_{\mathbf{y}} = \mathbf{s}_{\mathbf{y}}+\mathbf{t}_{\mathbf{y}}, \\ \mathbf{s}*\mathbf{t} &\text{ is defined by } (\mathbf{s}*\mathbf{t})_{\mathbf{y}} = \mathbf{s}_{\mathbf{y}}*\mathbf{t}_{\mathbf{y}}, \text{ and} \\ \mathbf{s}\vee\mathbf{t} &\text{ is defined by } (\mathbf{s}\vee\mathbf{t})_{\mathbf{y}} = \mathbf{s}_{\mathbf{y}}\vee\mathbf{t}_{\mathbf{y}}. \end{aligned} \tag{50}$$

Addition and multiplication are also defined for complex-valued templates. The arithmetic of extended real-valued templates is described in Reference 54.

Likewise, the operations of generalized convolution, as well as additive and multiplicative maximum, generalize to operations between templates. If $\mathbf{t}$ is a real- or complex-valued template from $\mathbf{Y}$ to $\mathbf{X}$, and $\mathbf{s}$ is a real- or complex-valued template from $\mathbf{X}$ to $\mathbf{W}$, the template $\mathbf{r} = \mathbf{s} \oplus \mathbf{t}$ from $\mathbf{Y}$ to $\mathbf{W}$ is defined by the image function $\mathbf{r_y}$, as follows

$$\mathbf{r_y}(\mathbf{w}) = \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{t_y}(\mathbf{x}) \cdot \mathbf{s_x}(\mathbf{w}), \text{ where } \mathbf{w} \in \mathbf{W}. \tag{51}$$

Computation of the weights $\mathbf{r_y}(\mathbf{w})$ can be accomplished by summing over a subset of $\mathbf{X}$. For example, given a point $\mathbf{y}$ in $\mathbf{X}$, then for each $\mathbf{w} \in \mathbf{W}$, we define the set $S(\mathbf{w}) = \{x \in X: \mathbf{x} \in S(\mathbf{t_y}) \text{ and } \mathbf{w} \in S(\mathbf{s_x})\}$. Then, since $\mathbf{t_y}(\mathbf{x}) \cdot \mathbf{s_x}(\mathbf{w}) = 0$ if $\mathbf{x} \notin S(\mathbf{w})$, we obtain

$$\mathbf{r_y}(\mathbf{w}) = \sum_{\mathbf{x} \in S(\mathbf{w})} \mathbf{t_y}(\mathbf{x}) \cdot \mathbf{s_x}(\mathbf{w}), \tag{52}$$

where the definition $\sum_{\mathbf{x} \in S(\mathbf{w})} \mathbf{t_y}(\mathbf{x}) \cdot \mathbf{s_x}(\mathbf{w}) = 0$ holds whenever $S(\mathbf{w}) = \varnothing$. The operations of additive and multiplicative maximum are similarly defined, and are discussed in detail in Reference 54.

Template composition and decomposition are useful for algorithmic optimization, which is the primary motivation for introducing operations between generalized templates. For example, if $\mathbf{r} = (1\ \mathit{1}\ 1)$ and $\mathbf{s} = (1\ \mathit{1}\ 1)'$ where the target point is italicized, and $\mathbf{r} = \mathbf{s} \oplus \mathbf{t}$, then the computation of $\mathbf{a} \oplus \mathbf{r} = \mathbf{a} \oplus (\mathbf{s} \oplus \mathbf{t})$ by $(\mathbf{a} \oplus \mathbf{s}) \oplus \mathbf{t}$ uses six local multiplications instead of nine. It has been shown that, if $\mathbf{r}$ is an $n \times n$ template, and $\mathbf{s}$ and $\mathbf{t}$ are the decompositions of $\mathbf{r}$ into $1 \times n$ and $n \times 1$ templates, respectively, then the computation of $\mathbf{a} \oplus \mathbf{r}$ by $(\mathbf{a} \oplus \mathbf{s}) \oplus \mathbf{t}$ uses $2n$ multiplications, instead of n^2. This concept is further discussed in Reference 55.

2.1.6. Recursive image algebra

Our summary of recursive IA operations (after Li [57]) includes an overview of recursive templates (Section 2.1.6.1), operations between images and recursive templates (Section 2.1.6.2), and operations between recursive templates (Section 2.1.6.3).

2.1.6.1. Recursive templates. A generalized recursive template is defined in terms of the generalized template of Section 2.1.3, together with a partial order imposed upon certain point sets.

A *partially ordered set* or *poset* P together with a binary relation $\prec$ satisfies the axioms of reflexivity, antisymmetry, and transitivity. Let $(P,\prec)$ be a poset and $[n] = \{1,2,...,n\}$, where $n = |P|$. A *linear extension of* $(P,\prec)$ is a bijection $\sigma\colon P \rightarrow [n]$ such that $(x \prec y) \in P \iff (\sigma(x) \leq \sigma(y)) \in [n]$.

Denote value set $\mathbf{F}$, point sets $\mathbf{X}$ and $\mathbf{Y}$, where $\prec$ is a partial order imposed upon $\mathbf{Y}$. A *generalized* $\mathbf{F}$*-valued recursive template from* $\mathbf{Y}$ *to* $\mathbf{X}$ is a function $\mathbf{t} = (\mathbf{t}_{\not\prec}, \mathbf{t}_{\prec})\colon \mathbf{Y} \rightarrow (\mathbf{F}^{\mathbf{X}}, \mathbf{F}^{\mathbf{Y}})$, which can be decomposed into a nonrecursive part $\mathbf{t}_{\not\prec}\colon \mathbf{Y} \rightarrow \mathbf{F}^{\mathbf{X}}$ and a recursive part $\mathbf{t}_{\prec}\colon \mathbf{Y} \rightarrow \mathbf{F}^{\mathbf{Y}}$, such that (a) $\mathbf{y} \notin \mathcal{S}(\mathbf{t}_{\prec}(\mathbf{y}))$, and (b) for each $\mathbf{z} \in \mathcal{S}(\mathbf{t}_{\prec}(\mathbf{y}))$, $\mathbf{z} \prec \mathbf{y}$. Thus, the support $\mathcal{S}(\mathbf{t}_{\prec}(\mathbf{y}))$ is consistent with the partial order imposed on $\mathbf{Y}$. As a result, for each $\mathbf{y} \in \mathbf{Y}$, $\mathbf{t}_{\not\prec}(\mathbf{y}) \in \mathbf{F}^{\mathbf{X}}$ and $\mathbf{t}_{\prec}(\mathbf{y}) \in \mathbf{F}^{\mathbf{Y}}$. For purposes of brevity, we denote $\mathbf{t}_{\not\prec\mathbf{y}} \equiv \mathbf{t}_{\not\prec}(\mathbf{y})$, $\mathbf{t}_{\prec\mathbf{y}} \equiv \mathbf{t}_{\prec}(\mathbf{y})$, and $\mathbf{t}_{\mathbf{y}} = (\mathbf{t}_{\not\prec\mathbf{y}}, \mathbf{t}_{\prec\mathbf{y}})$.

The consistency of $\mathbf{t}_{\prec}$ ensures the recursive computability of $\mathbf{t}$. Since a recursive template can be consistent with many partial orders on $\mathbf{Y}$ or any of their linear extensions, a recursive template that is consistent with a partial order $\prec$ is said to *admit* the partial order $\prec$. Thus, the set of all $\mathbf{F}$-valued recursive templates from $\mathbf{Y}$ to $\mathbf{X}$ that admits the partial order $\prec$ on $\mathbf{Y}$ is denoted by $(\mathbf{F}^{\mathbf{X}}, \mathbf{F}^{\mathbf{X}})^{\mathbf{Y}} \equiv (\mathbf{F}^{\mathbf{X}}, \mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}_{\prec}$. If $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}}, \mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$, then the support $\mathcal{S}(\mathbf{t}_{\mathbf{y}}) = (\mathcal{S}(\mathbf{t}_{\not\prec\mathbf{y}}), \mathcal{S}(\mathbf{t}_{\prec\mathbf{y}}))$. If $\mathcal{S}(\mathbf{t}_{\prec\mathbf{y}}) = \varnothing$, $\forall\, \mathbf{y} \in \mathbf{Y}$, then $\mathbf{t}$ collapses to $\mathbf{t}_{\not\prec\mathbf{y}}$. Thus, the following definitions can be specialized to nonrecursive templates by setting $\mathcal{S}(\mathbf{t}_{\prec\mathbf{y}}) = \varnothing$, $\forall\, \mathbf{y} \in \mathbf{Y}$.

A recursive template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}}, \mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ is *translation-invariant* if and only if for each triple $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{X}$ with $\mathbf{y}+\mathbf{z} \in \mathbf{X}$ and $\mathbf{x}+\mathbf{z} \in \mathbf{X}$, we have $\mathbf{t}_{\mathbf{y}}(\mathbf{x}) = \mathbf{t}_{\mathbf{y}+\mathbf{z}}(\mathbf{x}+\mathbf{z})$, which is equivalent to $\mathbf{t}_{\not\prec \mathbf{y}}(\mathbf{x}) = \mathbf{t}_{\not\prec \mathbf{y}+\mathbf{z}}(\mathbf{x}+\mathbf{z})$ and $\mathbf{t}_{\prec \mathbf{y}}(\mathbf{x}) = \mathbf{t}_{\prec \mathbf{y}+\mathbf{z}}(\mathbf{x}+\mathbf{z})$.

2.1.6.2. Operations between images and recursive templates. Recursive operations in image algebra are generalized in terms of the recursive right and left products, similar to Equations 43 and 45. Let Γ denote the global reduce operation, as before, and let $\gamma\colon \mathbf{F} \times \mathbf{F} \to \mathbf{F}$ be associative and commutative. Let $\mathbf{F}_1$ and $\mathbf{F}_2$ be two value sets with operations $\circ_1, \circ_2 : \mathbf{F}_1 \times \mathbf{F}_2 \to \mathbf{F}$. If $\mathbf{a} \in (\mathbf{F}_1)^{\mathbf{X}}$ and $\mathbf{t} \in (\mathbf{F}_2^{\mathbf{X}}, \mathbf{F}_2^{\mathbf{Y}})_{\prec}^{\mathbf{Y}}$, then the *generalized recursive right product* is the binary operation $\textcircled{\Gamma}_{\prec}\colon \mathbf{F}_1^{\mathbf{X}} \times (\mathbf{F}_2^{\mathbf{X}}, \mathbf{F}_2^{\mathbf{X}})^{\mathbf{Y}} \to \mathbf{F}^{\mathbf{Y}}$, defined as

$$\mathbf{a} \textcircled{\Gamma}_{\prec} \mathbf{t} \equiv \left\{ (\mathbf{y}, \mathbf{c}(\mathbf{y}))\colon \mathbf{c}(\mathbf{y}) = \underset{\mathbf{x} \in \mathcal{S}(\mathbf{t}_{\not\prec \mathbf{y}})}{\Gamma} \mathbf{a}(\mathbf{x}) \circ_1 \mathbf{t}_{\not\prec \mathbf{y}}(\mathbf{x}) \; \gamma \underset{\mathbf{z} \in \mathcal{S}(\mathbf{t}_{\prec \mathbf{y}})}{\Gamma} \mathbf{c}(\mathbf{z}) \circ_2 \mathbf{t}_{\prec \mathbf{y}}(\mathbf{z}),\ \mathbf{y} \in \mathbf{Y} \right\}, \tag{53}$$

with the left product symmetrically defined.

For example, if $\mathbf{F}, \mathbf{F}_1, \mathbf{F}_2 \subset \mathbf{R}$, $\gamma = +$, and $\circ_1, \circ_2 = \cdot$, then $\textcircled{\Gamma}_{\prec} = \bigoplus_{\prec}$, the recursive convolution. Likewise, if $\mathbf{F}, \mathbf{F}_1, \mathbf{F}_2 = \mathbf{R}_{-\infty}$, $\gamma = \vee$, and $\circ_1, \circ_2 = +$, then $\textcircled{\Gamma}_{\prec} = \boxbar_{\prec}$, the recursive additive maximum.

2.1.6.3. Operations between recursive templates. It is well established that image algebra template composition and decomposition are useful for algorithmic optimization. For example, Li showed that for $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ and for $\mathbf{s}, \mathbf{t} \in (\mathbf{R}_{-\infty}^{\mathbf{X}})^{\mathbf{Y}}$,

$$\begin{aligned} \mathbf{a} \oplus (\mathbf{s} \oplus \mathbf{t}) &= (\mathbf{a} \oplus \mathbf{s}) \oplus \mathbf{t} \quad \text{and} \\ \mathbf{a} \oplus (\mathbf{s} + \mathbf{t}) &= (\mathbf{a} \oplus \mathbf{s}) + (\mathbf{a} \oplus \mathbf{t}). \end{aligned} \tag{54}$$

Additionally, if $\mathbf{a} \in \mathbf{R}_{-\infty}^{\mathbf{X}}$ and $\mathbf{s}, \mathbf{t} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{Y}}$, then

$$\begin{aligned} \mathbf{a} \boxbar (\mathbf{s} \boxbar \mathbf{t}) &= (\mathbf{a} \boxbar \mathbf{s}) \boxbar \mathbf{t} \quad \text{and} \\ \mathbf{a} \boxbar (\mathbf{s} \vee \mathbf{t}) &= (\mathbf{a} \boxbar \mathbf{s}) \vee (\mathbf{a} \boxbar \mathbf{t}). \end{aligned} \tag{55}$$

Li [57] extended template decomposition to recursive templates, and proved certain associative and distributive laws. We summarize salient theory, as follows.

Let the finite point sets $\mathbf{X}, \mathbf{W} \subset \mathbf{R}^n$ and let the recursive templates $\mathbf{t} \in (\mathbf{R}^{\mathbf{X}}, \mathbf{R}^{\mathbf{X}})^{\mathbf{X}}_{\prec}$ and $\mathbf{s} \in (\mathbf{R}^{\mathbf{W}}, \mathbf{R}^{\mathbf{X}})^{\mathbf{X}}_{\prec}$. The generalized recursive convolution of two recursive templates, denoted by $\mathbf{r} = \mathbf{s} \bigoplus_{\prec} \mathbf{t}$, where $\mathbf{r} = (\mathbf{r}_{\not\prec}, \mathbf{r}_{\prec}) \in (\mathbf{R}^{\mathbf{W}}, \mathbf{R}^{\mathbf{X}})^{\mathbf{X}}_{\prec}$, is defined as

$$\mathbf{r}_{\not\prec} = \mathbf{s}_{\not\prec} \bigoplus \mathbf{t}_{\not\prec} \quad \text{and} \quad \mathbf{r}_{\prec} = 1 - (1 - \mathbf{s}_{\prec}) \bigoplus (1 - \mathbf{t}_{\prec}), \tag{56}$$

where $1 \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$ denotes the unitary template, which is defined in terms of its weights as

$$1_{\mathbf{x}}(\mathbf{z}) = \begin{cases} 1 \text{ if } \mathbf{z} = \mathbf{x} \\ 0 \text{ otherwise} \end{cases}, \quad \mathbf{x}, \mathbf{z} \in \mathbf{X}. \tag{57}$$

Given the previously-defined templates $\mathbf{s}$ and $\mathbf{t}$, the recursive addition $\mathbf{r} = \mathbf{s} +_{\prec} \mathbf{t}$ is given by

$$\begin{aligned} \mathbf{r}_{\not\prec} &= \mathbf{s}_{\not\prec} \bigoplus (1 - \mathbf{t}_{\prec}) + (1 - \mathbf{s}_{\prec}) \bigoplus \mathbf{t}_{\not\prec} \\ \mathbf{r}_{\prec} &= 1 - (1 - \mathbf{s}_{\prec}) \bigoplus (1 - \mathbf{t}_{\prec}). \end{aligned} \tag{58}$$

Nonlinear recursive template composition and inversion are defined in Reference 57. Having concluded the discussion of image algebra, we next consider notation specific to this document.

2.2. Study Notation

In addition to image algebra, which comprises the majority of the notation employed in this study, we adopt the following conventions:

a) Image transformations are expressed in general form as T: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, where $\mathbf{F} \neq \mathbf{G}$ and $\mathbf{X} \neq \mathbf{Y}$ are possible.

b) Operations in *image space*, i.e., over *domain*(T), are denoted by $\bigcirc$, unless otherwise stated.

c) Operations over *range*(T) are usually denoted by $\bigcirc'$.

d) The images of primary concern are denoted by

   a the source image,

$\mathbf{a}_c$ the transformed image $T(\mathbf{a})$,

$\hat{\mathbf{a}}$ the result of the operation $\bigcirc(\mathbf{a})$, and

$\hat{\mathbf{a}_c}$ the result of the operation $\bigcirc'(\mathbf{a}_c)$.

e) The i-th derivative of an image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ is denoted as

$$\frac{d^i(\mathbf{a})}{d\,\mathbf{x}^i}, \mathbf{x} \in \mathbf{X}, \tag{59}$$

provided that the operation d is defined over $\mathbf{X}$. Assuming that $\mathbf{X} \subset \mathbf{R}^n$, the i-th partial derivative with respect to the projection of a point $\mathbf{x} \in \mathbf{X}$ is denoted as

$$\frac{\partial^i(\mathbf{a})}{\partial(p_k(\mathbf{x}))^i}. \tag{60}$$

f) In addition to the sets described in Section 2.1.1, which are denoted in open face ($\mathbf{F}$) and bold face ($\mathbf{X}$) type, sets may also be specified by upper-case letters in normal (S), italic (G), script ($\mathcal{P}$), or Gothic ($\mathfrak{D}$) faces.

g) The properties of a set S are usually denoted by $\mathcal{P}$(S), unless otherwise stated.

h) Set properties that are disclosed or concealed from the scrutiny of an oracle $\mathfrak{O}$ (defined in Chapter 5) are denoted by $\mathcal{D}$(S) and $\mathcal{H}$(S), respectively.

i) Scalar variables are denoted in lower case by normal face type (e.g., x, y) or by italic type (e.g., x, y), unless otherwise stated.

j) The operator $(\cdot)$ denotes scalar multiplication when applied to scalar variables. When $(\cdot)$ is applied to one- or two-dimensional matrices, or to images defined on a subset of $\mathbf{R}$ or $\mathbf{R}^2$, then matrix multiplication is indicated. In contrast, Hadamard multiplication of matrices or images is denoted by (*).

k) The time complexity of an operation f is denoted by T(f), which is not to be confused with the transform notation T (italic font). Similarly, the space complexity of f is denoted by S(f), and the work required to compute f, by W(f). Note that the work W is not to be confused with $\mathbf{W}$, which denotes a point set. Additionally, the cost of computing f is denoted by C(f), which is not to be confused with C, an

image transform that is used in Chapter 4. When we compare the computation of f over n input data on various N-processor architectures A, then we occasionally elaborate T, S, W, and C as $T_{A(f,N)}(n)$.

1) The superscript asterisk (*) may denote template conjugation (e.g., $\mathbf{t}^*$), complex conjugation of an image (e.g., $\mathbf{a}^*$), or may denote an approximation to an inverse transform (e.g., T^*), which is defined in the following chapter.

CHAPTER 3
FUNDAMENTAL THEORY

In this chapter, we develop theory that describes the processing of transformed data. Section 3.1 contains a review of the concepts of homomorphism, isomorphism, and heteroassociativity. In Section 3.2 we present the basic theory of computation over compressed or encrypted imagery, which is related to several problems of computer science, such as the transform properties problem. Section 3.3 contains a taxonomy of compressive and encryptive transformations that is organized in four classes. Such classification facilitates the development of high-level methods for deriving functions that compute over the range spaces of transforms in each class, as exemplified in Section 3.4.

3.1. Mathematical Concepts.

This study is based in part upon the concepts of commutativity diagrams, homomorphism, and isomorphism. Homomorphic transformations facilitate the processing of transformed imagery in a manner analogous to operations that process the source image. An isomorphism is a special case of a homomorphism. Isomorphisms are important in the direct mapping of operations over one domain to operations in another domain. Additionally, isomorphism is useful when discussing data security, as in Chapter 4. We begin with a summary of basic mathematical terms.

3.1.1. Basic Theory.

The concepts of elements, sets, and mappings are key to our theoretical development, and constitute the basic building blocks of image algebra.

3.1.1.1. Definition. The *n-fold Cartesian product of a set* S is defined by

$$S^n \equiv \prod_{i=1}^{n} S = \{(s_1, s_2, \ldots, s_n) : s_1, s_2, \ldots, s_n \in S\}. \tag{61}$$

3.1.1.2. Definition. A *mapping* M *from set* R *to set* S is denoted by M: R $\rightarrow$ S. The collection of all mappings R $\rightarrow$ S is denoted by S^R. Thus, if M: R $\rightarrow$ S, then M $\in S^R$. We call R the *domain of* M, written as R = *domain*(M). The *range* of M, is defined as *range*(M) = $\{s \in S : s = M(r)$ for some $r \in R\}$.

3.1.1.3. Definition. An m-*to*-k-*ary operation* $\bigcirc$ *from set* R *to set* S is a mapping $\bigcirc : \prod_{i=1}^{m} R \rightarrow \prod_{j=1}^{k} S$, where it is hereafter understood that m,k ≥ 1. When k=1, we call $\bigcirc$ an *m-ary operation*, denoted by $\bigcirc_m$.

3.1.2. Groupoids, Operations, and Transformations.

We next state facts concerning groupoids, operations, and transformations, which lead to definitions of homomorphism and isomorphism. Groupoids combine the concepts of sets and operations, and lead to the consideration of transforms and their inverses. The definition of a transformation leads to discussion of homomorphism and isomorphism.

3.1.2.1. Definition. A *groupoid* $(S,\bigcirc)$ is comprised of the set S and a binary operation $\bigcirc : S \times S \rightarrow S$. A groupoid whose binary operation is associative is called a *semigroup.*

3.1.2.2. Definition. We call e *the identity element of* S *with respect to the operation* $\bigcirc : S \times S \rightarrow S$, if and only if

$$f \bigcirc e = e \bigcirc f = f, \forall f \in S. \tag{62}$$

3.1.2.3. Definition. Let S be a set with identity e with respect to the operation $\bigcirc: S \times S \to S$. If $f,g \in S$ then g is called the *inverse of* $f \in S$ *with respect to the operation* $\bigcirc$, if and only if

$$f \bigcirc g = g \bigcirc f = e, \forall f \in S. \tag{63}$$

If $f \in S$ and $g \in S$ is the inverse of f, then we denote g by f^{-1}.

3.1.2.4. Definition. Suppose $T: S \to S'$ and $V: S' \to S$. Then V is called the *inverse* of the transformation T if and only if

$$(V \circ T)(f) = f, \forall f \in S. \tag{64}$$

If $T: S \to S'$ and $V: S' \to S$ is the inverse of T, then we denote V by T^{-1}. If we denote $T(f)$ by f' (i.e., $f' \equiv T(f)$), then it follows that $T^{-1}(f') = f, \forall f \in S$.

3.1.2.5. Definition. Let $T: S \to S'$ be a homomorphism. If S is a normed space with norm $\| \ \|$ and $\epsilon \in \mathbf{R}$, then we say that a homomorphism $T^*: \mathbf{G}^{\mathbf{Y}} \to \mathbf{F}^{\mathbf{X}}$ is an *ϵ-near inverse approximation of* T if and only if

$$\| T^*(T(\mathbf{a})) - \mathbf{a} \| < \epsilon, \ \forall \mathbf{a} \in \mathbf{F}^{\mathbf{X}}. \tag{65}$$

Note that every inverse is an ϵ-near inverse for all $\epsilon > 0$.

3.1.2.6. Definition. Given groupoids $(S,\bigcirc)$ and $(S',\bigcirc')$, the transformation $T: S \to S'$ is a *homomorphism from* S *to* S' if and only if

$$T(f \bigcirc g) = T(f) \bigcirc' T(g), \forall f,g \in S. \tag{66}$$

3.1.2.7. Definition. Let $(S,\bigcirc)$ and $(S',\bigcirc')$ be groupoids with homomorphism $T: S \to S'$. Then, $T^{-1}: S' \to S$ is an *inverse homomorphism for* T if and only if T^{-1} is the inverse transform of T and T^{-1} is a homomorphism.

3.1.2.8. Definition. If $T: S \rightarrow S'$ is a homomorphism and T is a one-to-one and onto mapping, then T is an *isomorphism of* S *onto* S'.

3.1.2.9. Definition. A *semigroup* is a groupoid whose binary operation is associative.

3.1.2.10. Definition. A *monoid* is a semigroup with identity.

3.1.2.11. Theorem. Let $(S,\bigcirc)$ and $(S',\bigcirc')$ be monoids, and let e denote the identity of S. If $T: S \rightarrow S'$ is an isomorphism, then the following statements hold:

(i) If e' is the identity element of S', then $e' = T(e)$, and

(ii) $T(f^{-1}) = (T(f))^{-1}$, $\forall f \in S$ which has an inverse $f^{-1} \in S$.

Proof. The proof is well known. □

3.1.3. Preservation of Set Properties.

We next develop theory that describes the preservation of various properties of sets and mappings under homomorphism. The following theory is fundamental to the derivation of functions that compute over the range space of an image transformation.

3.1.3.1. Notation. Denote the properties of a collection of subsets of a set S by $\mathcal{P}(S)$. If $A \subset S$ has property q and A is a singleton set (i.e., $A = \{u\}$), then we say that u has property q.

3.1.3.2. Definition. We say that the transform $T: S \rightarrow S'$ *preserves the properties* $\mathcal{P}(S)$ if and only if for each property $q \in \mathcal{P}(S)$ and for all $A \subset S$ with property q, $T(A)$ also has property q.

We illustrate this concept with the following examples.

3.1.3.3. Example. Let $q \in \mathcal{P}(S)$ be the property of connectivity, and let $T: S \rightarrow S'$ preserve q. This means that for every connected set $A \subset S$, $T(A)$ must also be connected.

3.1.3.4. Example. Given an image $\mathbf{a} \in (\mathbb{Z}_n)^{\mathbf{X}}$, it is easily verified that the sorted histogram $\mathbf{h}(\mathbf{a})$ is preserved under the transformation $f(\mathbf{a}) = n - \mathbf{a}$.

3.1.3.5. Example. The fact that associativity and commutativity are preserved under homomorphisms follows from the fact that the operations of algebraic structures are preserved under homomorphisms.

3.1.3.6. Definition. We say that a binary operation $\bigcirc : S \times S \rightarrow S$ *preserves property* $q \in \mathcal{P}(S)$ if and only if, for all pairs of subsets $A, B \subset S$ with property q, or for each pair of elements $f, g \in S$ with property q, then $\bigcirc(A,B)$ or $f \bigcirc g$ exhibits property q.

3.1.3.7. Definition. A groupoid $(S, \bigcirc)$ is said to *exhibit property* q if the operation $\bigcirc$ preserves property q.

3.1.3.8. Definition. An *algebra* $\mathcal{A} = (\mathcal{F}, \mathcal{O})$ consists of a set $\mathcal{F}$ of *operands* and a set $\mathcal{O}$ of *operations* upon the operands in $\mathcal{F}$.

3.1.3.9. Example. The algebra consisting of the real numbers together with the operations of multiplication and addition is denoted by the tuple $(\mathbf{R}, \{+, \cdot\})$, which can be written as $(\mathbf{R}, +, \cdot)$ for purposes of simplicity.

3.1.4. Heteroassociativity.

We next discuss the property of heteroassociativity, which is employed in the derivation of operations over the range spaces of certain encryptive transforms.

3.1.4.1. Definition. An operation $\gamma : S \times S \rightarrow S$ is *associative with respect to the operation* $\bigcirc : S \times S \rightarrow S$, if and only if

$$(f \bigcirc g) \gamma h = f \bigcirc (g \gamma h), \forall f, g, h \in S. \tag{67}$$

3.1.4.2. Definition. If an operation γ is associative with respect to an operation $\bigcirc$, and $\bigcirc$ is associative with respect to γ, then we say that the operations $\gamma, \bigcirc : S \times S \to S$ are *heteroassociative.*

3.1.4.3. Definition. An algebra $(S, \bigcirc, \gamma)$ is called a *heteroassociative algebra* if and only if the operations $\gamma, \bigcirc : S \times S \to S$ are heteroassociative.

3.1.5. Heteroassociativity of Inverse Functions.

We next consider inverse operations and their heteroassociative properties.

3.1.5.1. Definition. Let $\bigcirc : S \times S \to S$ and let $S' = \{f \in S: \text{ f has an inverse}\}$. Then, $\Omega : S \times S' \to S$ is called the *inverse operation* (or *inverse*) *of* $\bigcirc$ if and only if

$$f \bigcirc g^{-1} = f \Omega g, \forall f \in S \text{ and } \forall g \in S', \tag{68}$$

where g^{-1} denotes the inverse of g. If Ω is the inverse operation of $\bigcirc$, then we denote Ω by $\bigcirc^{-1}$.

3.1.5.2. Remark. In the preceding definition, $\bigcirc^{-1}$ does not mean the *inverse function* $S \to S \times S$. It should be clear from the context when we mean inverse operation versus inverse function.

3.1.5.3. Example. Referring to the preceding definition, note that if $S = \mathbf{R}$, then $S' = S$ for the operation of addition. Thus, subtraction is the inverse of addition over the real numbers. In contrast, although division is the inverse of multiplication over the real numbers, $S' = \mathbf{R} \setminus \{0\} \subset \mathbf{R}$, since $f/0$ remains undefined for all f in $\mathbf{R}$.

3.1.5.4. Definition. Assume that S is a normed space with norm $\| \ \|$. Let $\epsilon \in \mathbf{R}^+$ and $S^* = S \times S$. An operation $\bigcirc : S^* \to S$ has an *ϵ-near inverse approximation* if and only if

$$f \bigcirc g^{-1} = f \bigcirc^* g, \forall g \in S', \tag{69}$$

where S' was defined in Section 3.1.5.1 and

$$\|(f \bigcirc^* f) - e\| < \epsilon, \forall f \in S, \tag{70}$$

with e denoting the identity of S.

3.1.5.5. Observation. Assume that $(S,\bigcirc)$ and $(S',\bigcirc')$ are two monoids, where operations $\bigcirc : S \times S \to S$ and $\bigcirc' : S' \times S' \to S'$ each have an inverse denoted by $\bigcirc^{-1} : S \times S \to S$ and $(\bigcirc')^{-1} : S' \times S' \to S'$. If $T : S \to S'$ is a homomorphism, then $T(f \bigcirc f^{-1}) = T(f)(\bigcirc')^{-1} T(f), \forall f^{-1} \in S$.

3.1.5.6. Definition. A monoid $(S,\bigcirc)$ is called a *group* if and only if each element $f \in S$ has an inverse $f^{-1} \in S$. Alternatively, we may define a group as a monoid with identity and an inverse, whose binary operation is associative. If $\bigcirc$ is commutative, then the group $(S,\bigcirc)$ is called an *Abelian group.*

3.1.5.7. Definition. A *ring* $(S,\bigcirc,\gamma)$ satisfies the following conditions for all $f,g,h \in S$:

(i) The functions $\bigcirc, \gamma : S \times S \to S$ are associative and commutative.

(ii) γ has an identity $e \in S$ and there exists $\forall f \in S$ an inverse f^{-1} such that $f \gamma f = e$.

(iii) $\bigcirc$ is left- and right-distributive over γ, i.e., $f \bigcirc (g \gamma h) = (f \bigcirc g) \gamma (f \bigcirc h)$ and $(f \gamma g) \bigcirc h = (f \bigcirc h) \gamma (g \bigcirc h), \forall f,g,h \in S$.

3.1.5.8. Definition. A *semi-ring* $(S,\bigcirc)$ is a commutative semigroup. Note that this is different from an Abelian group $(S,\bigcirc)$ that is present in a ring $(S,\bigcirc,\gamma)$.

3.1.5.9. Lemma. Let $(S,\bigcirc)$ be a group. If $\bigcirc$ is associative, then the following statements hold:

(i) $\bigcirc^{-1}$ is associative, and

(ii) $(f \bigcirc g) \bigcirc^{-1} h = f \bigcirc (g \bigcirc^{-1} h)$, $\forall f,g,h \in S$, where $\bigcirc^{-1}$ denotes the inverse operation of $\bigcirc$.

Proof. The proof follows from the definitions of a group and an inverse function. □

3.1.5.10. Definition. Suppose that $\mathcal{A} = (S,\bigcirc,\gamma)$ and $\mathcal{A}' = (S',\bigcirc',\gamma')$ are algebras, where $\bigcirc$ and γ are binary operations over S, and $\bigcirc'$ and γ' are binary operations over S'. A function $T: S \rightarrow S'$ is called a *homomorphism from algebra $\mathcal{A}$ to algebra $\mathcal{A}'$* if and only if the following conditions are satisfied for all $f,g \in S$:

(i) $T(f \bigcirc g) = T(f) \bigcirc' T(g)$, and

(ii) $T(f \gamma g) = T(f) \gamma' T(g)$.

Note that the correspondence of operations $\bigcirc$ and $\bigcirc'$, as well as γ and γ', must be preserved. It follows that the cardinality of the set of operations in algebra $\mathcal{A}$ must equal the cardinality of the set of operations in $\mathcal{A}'$.

This concludes the definitions upon which our theory of processing compressed data is based. We introduce the core theory in the following section.

3.2. Image Compression, Encryption, and Compressive Computation.

A mathematical discussion of compressive computation requires the following definitions as background. Section 3.2.1 contains definitions of parameters that characterize compressive transforms. In Section 3.2.2, we define operations that process over the range space of such transforms.

3.2.1. Parameters of Compressive Transforms.

3.2.1.1. Definition. Given a finite set $S \subset \mathbb{N}$ and $r \in \mathbb{N}$, the *maximum field width of* S, denoted by $siz(S)$, is the maximum number of radix-r digits required to encode a value in S.

3.2.1.2. Example. If finite $S \subset \mathbb{R}^+$ is quantized under an optimal space constraint, then the binary representation (r=2) of S has $siz(S) = \lceil \log_2(|S|) \rceil$. However, optimality does not necessarily imply high accuracy, since error can result when elements of S are quantized. The following example is illustrative.

3.2.1.3. Example. If finite $S \subset \mathbb{R}^+$ is quantized linearly into a radix-r representation using a quantization interval $q \in \mathbb{R}^+$, then $siz(S)$ is bounded as

$$\lceil \log_r |S| \rceil \leq siz(S) \leq \vee\{y : y = \lceil \log_r(x/q) \rceil, x \in S\}, \tag{71}$$

where q denotes quantization interval width and the inequality accounts for data that may not be normalized to the interval $[\wedge S, \vee S]$.

3.2.1.4. Definition. An *encoding transform* is a mapping $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, where $\mathbf{X} \neq \mathbf{Y}$ and $\mathbf{F} \neq \mathbf{G}$ are possible.

3.2.1.5. Definition. An *exact encoding* $\mathbf{a}_c$ of an image $\mathbf{a}$ by a transform T fulfills the following conditions:

(i) No additional information is introduced into *domain*(T). Implementationally, this means that pixels are not added to $\mathbf{a}$ prior to applying T.

(ii) No information is lost in the encoding. That is, T encodes all the information in $\mathbf{a}$ such that the information in $\mathbf{a}_c$ completely describes the information in $\mathbf{a}$.

(iii) No information is added to the encoded result. For example, $\mathbf{a}_c$ is not augmented with additional pixels to fulfill a given format specification.

Exact encodings are often called *information-preserving* or *lossless* transformations. If T is not exact, then it is called *inexact.* Alternative terms are *information-nonpreserving* or *lossy.*

3.2.1.6. Definition. The *domain compression ratio* of a transform T: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is given by

$$\mathrm{CR_D}(T) = \frac{|\mathbf{X}|}{|\mathbf{Y}|}, \tag{72}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are finite point sets.

3.2.1.7. Definition. The *range compression ratio* of a transform T: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is given by

$$\mathrm{CR_R}(T) = \frac{siz(\mathbf{F})}{siz(\mathbf{G})}, \tag{73}$$

where implementations of $\mathbf{F}$ and $\mathbf{G}$ would be finite sets.

3.2.1.8. Definition. The *compression ratio* of a transform T: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is given by

$$\mathrm{CR}(T) = \frac{|\mathbf{X}| \cdot siz(\mathbf{F})}{|\mathbf{Y}| \cdot siz(\mathbf{G})}. \tag{74}$$

It follows from the preceding definitions that $\mathrm{CR} = \mathrm{CR_R} \cdot \mathrm{CR_D}$. Thus, CR describes compression at a digit (bit) level for radix-r (binary) encodings, under the assumption of constant field width. This assumption is reasonable for machines with fixed-size registers, which include the majority of digital computers.

3.2.1.9. Definition. A *compressive transform* T: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is an encoding transform whose compression ratio $\mathrm{CR}(T) > 1$.

Note that a compressive transform T may be a homomorphism or an isomorphism, which leads to the following definitions of operations over *range*(T).

3.2.2. Transform-regime Analogues, Duals, and Computational Systems.

The concept of computing over compressed (encrypted) imagery, which we call *compressive (encryptive) processing*, can be illustrated by the commutativity diagram shown in Figure 1. Here, $\bigcirc$ denotes a unary *image operation* that is applied to image **a** to yield image $\hat{\mathbf{a}}$. The transform T, which may be information-nonpreserving, maps image **a** to a *compressed image* $\mathbf{a}_c$, which is accepted by an operation $\bigcirc'$ that yields the image $\hat{\mathbf{a}}_c$. The operation $\bigcirc'$ is called a *transform-regime analogue* or *analogue* of $\bigcirc$. Similarly, an operation Ω that transforms $\mathbf{a}_c$ into $\hat{\mathbf{a}}$ is called a *dual* of $\bigcirc$.

If T has an inverse T^{-1} (or an approximation T^* to its inverse, in the case of lossy T), then the image $\hat{\mathbf{a}}_c$ can be transformed to yield an exact representation of (approximation to) $\hat{\mathbf{a}}$.

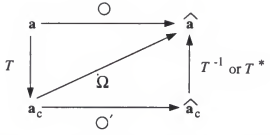


Figure 1. Commutativity diagram for processing compressed or encrypted imagery with unary operations.

Since it is occasionally possible that dual operations can be derived in the presence of intractable analogues, duals are useful in practice. We later show how to use analogues and duals to formulate computational systems based upon homomorphisms and isomorphisms. This prepares the reader for further theoretical development in Parts II and III.

3.2.2.1. Assumption. Let $(S, \bigcirc)$ and $(S', \bigcirc')$ be groups, and let $T: S \rightarrow S'$ be a homomorphism.

3.2.2.2. Definition. If Assumption 3.2.2.1 holds, then the operation $\bigcirc'$ is called *an* $\bigcirc$-*property of* T, or a *transform-regime analogue* of $\bigcirc$ over *range*(T).

3.2.2.3. Remark. We say that the determination of a specific property of a given transform is called the *Transform Properties Problem* (TPP). A theoretical discussion of this problem is given later in this section, where we show how the TPP can be used to construct compressive computational systems.

3.2.2.4. Example. The Fourier transform $\mathcal{F}\colon \mathbf{C}^{\mathbf{X}} \to \mathbf{C}^{\mathbf{X}}$ has a convolution property (i.e., $\bigcirc = \bigoplus$) that is given by $\bigcirc' = \star$, the operation of Hadamard multiplication. Alternatively, we say that * is a transform-regime analogue of $\bigoplus$ over *range*($\mathcal{F}$).

3.2.2.5. Definition. If Assumption 3.2.2.1 holds and $\Omega\colon S' \times S' \to S$ such that $\Omega(T(f), T(g)) = \bigcirc(f,g), \forall f,g \in S$, then Ω is called a *transform-regime dual* of $\bigcirc$, over *range*(T).

3.2.2.6. Definition. If Assumption 3.2.2.1 holds, T has an ϵ-near approximation to its inverse, and $\Omega\colon S' \times S' \to S$ such that $\|\Omega(T(f), T(g)) - \bigcirc(f,g)\| < \epsilon, \forall f,g \in S$, then Ω is called an *ϵ-near approximation to a transform-regime dual* of $\bigcirc$ over *range*(T). For purposes of brevity, we also call Ω an *approximate dual* of $\bigcirc$ over *range*(T).

3.2.2.7. Definition. If Assumption 3.2.2.1 holds, then the structure $((S,\bigcirc), T,(S',\bigcirc'))$ is called a *homomorphic computational system* (HCS).

3.2.2.8. Definition. If the transform T in Assumption 3.2.2.1 is an isomorphism, then the structure $\left((S,\bigcirc), T,(S',\bigcirc'), T^{-1}\right)$ is called an *isomorphic computational system* (ICS).

3.2.2.9. Definition. If Assumption 3.2.2.1 holds and $T\colon S \to S'$ has an ϵ-near approximation to its inverse, denoted by $T^*\colon S' \to S$, then the structure $((S,\bigcirc), T,(S',\bigcirc'), T^*)$ is called an *ϵ-near isomorphic computational system* (ϵ-ICS).

3.2.2.10. Definition. If Assumption 3.2.2.1 holds and T is a compressive transform, then the corresponding HCS (ICS) is said to be a *compressive HCS (ICS)*.

We can extend the definition of an HCS or ICS to include operations with multiple operands of different type, or multiple operations that each have a corresponding analogue. In order to do this, we first define several terms pertaining to algebras.

3.2.2.11. Definition. Given the indexed sets $\mathcal{F} = \{\mathbf{F}_\lambda\}$ and $\mathcal{G} = \{\mathbf{G}_\mu\}$, an algebra $\mathcal{B} = (\mathcal{G}, \mathcal{U})$ is a *subalgebra* of algebra $\mathcal{A} = (\mathcal{F}, \mathcal{O})$ if and only if

(i) $\mathcal{U} \subset \mathcal{O}$, and

(ii) For each $\mathbf{G}_\mu \in \mathcal{G}$, there exists an $\mathbf{F}_\lambda \in \mathcal{F}$ such that $\mathbf{G}_\mu \subset \mathbf{F}_\lambda$.

3.2.2.12. Definition. An algebra $\mathcal{A} = (\mathcal{F}, \mathcal{O})$ is a *homogeneous algebra* iff $|\mathcal{F}| = 1$ and is a *heterogeneous algebra* iff $|\mathcal{F}| > 1$.

3.2.2.13. Observation. Since a homomorphism T preserves the operations of an algebra, an HCS may be formulated using a set of operations $\mathcal{O}$ rather than a single operation. For example, consider the structure $((S, \mathcal{O}), T, (S', \mathcal{O}'))$, where $\mathcal{O}'$ contains the transform-regime analogues of the corresponding operations in $\mathcal{O}$. The symmetric case holds for ICS.

Of primary interest to this study is the determination of the $\mathcal{O}$-properties of T, where $\mathcal{O}$ are common image operations and T is a compressive transform. A more difficult problem, which address only tangentially, concerns the derivation of $\mathcal{O}$-properties of encryptive transforms.

A key theoretical constraint encountered early in this study is implied by the fact that homomorphisms are defined for binary operations only. Thus, HCS were initially defined for only one type of image algebra operation, namely, binary pointwise operations. In practice, this was an unacceptable restriction, since image algebra contains unary and binary pointwise operations as well as unary global-reduce operations and binary operations with heterogeneous operands (e.g., image-template and template-template operations). We

thus developed the following theory that unifies compressive computation over all image algebra operations by expressing the concept of an HCS in terms of a family of mappings.

3.2.2.14. Definition. Let algebras $\mathcal{A} = \{A_1, A_2, \ldots, A_n; \mathcal{O}\}$ and $\mathcal{B} = \{B_1, B_2, \ldots, B_n; \mathcal{O}'\}$, where $\mathcal{O}$ and $\mathcal{O}'$ denote sets of operations. For example, $A_1 = \mathbf{F}$, $A_2 = \mathbf{F}^{\mathbf{X}}$, $A_3 = (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, and $\mathcal{O} = \{+, \cdot, \oplus\}$. A *family of mappings* $\zeta = \{\zeta_1, \zeta_2, \ldots \zeta_n; \Theta\}$, where $\zeta_i : A_i \to B_i$, for i = 1..n, is *an algebraic homomorphism* $\zeta : \mathcal{A} \to \mathcal{B}$ if and only if, for each $\bigcirc \in \mathcal{O}$, (a) there exists an analogous operation $\bigcirc' \in \mathcal{O}'$ over *range*(ζ) and (b) $\Theta : \mathcal{O} \to \mathcal{O}'$, such that the following statement holds. If $\bigcirc_i : A_{i_1} \times A_{i_2} \times \cdots \times A_{i_m} \to A_{i_j}$, then $\Theta(\bigcirc) = \bigcirc' : B_{i_1} \times B_{i_2} \times \cdots \times B_{i_m} \to B_{i_j}$ such that

$$\Theta(\bigcirc)[\zeta_{i_1}(\mathbf{a}_{i_1}), \zeta_{i_2}(\mathbf{a}_{i_2}), \ldots, \zeta_{i_m}(\mathbf{a}_{i_m})] = \zeta_j[\bigcirc(\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \ldots, \mathbf{a}_{i_m})]. \tag{75}$$

3.2.2.15. Remark. If Definition 3.2.2.14 holds and $\Theta(\bigcirc) = \bigcirc'$ is a dual over *range*(ζ) of $\bigcirc$, then

$$\Theta(\bigcirc)[\zeta_{i_1}(\mathbf{a}_{i_1}), \zeta_{i_2}(\mathbf{a}_{i_2}), \ldots, \zeta_{i_m}(\mathbf{a}_{i_m})] = \bigcirc(\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \ldots, \mathbf{a}_{i_m}). \tag{76}$$

3.2.2.16. Observation. Given Definition 3.2.2.14 and T: $\mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{Y}}$, one encounters the following cases in nonrecursive image algebra:

Case 1. If m = 1, then the commutativity diagram of Figure 1 is satisfied, i.e., $\bigcirc'(T(\mathbf{a})) = T(\bigcirc(\mathbf{a}))$. For such unary operations, we form the following structure:

$$X_{\bigcirc} = (\mathcal{A}, \zeta, \mathcal{B}) = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \bigcirc'\}). \tag{77}$$

If T is a compressive transform, then we call X a *compressive computational system* (CCS).

Case 2. If m = 1, then let $A_1 = \mathbf{F}$, $A_2 = \mathbf{F}^{\mathbf{X}}$, and $\zeta = \{T, \Theta\}$ such that the global reduce operation $\Gamma : \mathbf{F}^{\mathbf{X}} \to \mathbf{F}$ has an analogue $\Theta(\bigcirc) = \Gamma' : \mathbf{G}^{\mathbf{Y}} \to \mathbf{G}$ for which Equation 75 holds. The corresponding CCS is given by

$$X_{\Gamma} = (\mathcal{A}, \zeta, \mathcal{B}) = (\{\mathbf{F}^{\mathbf{X}}, \mathbf{F}; \Gamma\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \mathbf{G}; \Gamma'\}). \tag{78}$$

Case 3. If m = 2 and $A_1 = A_2 = \mathbf{F}^{\mathbf{X}}$, then Equation 75 reduces to the definition of a homomorphism T. For example, if $\bigcirc: \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$, $\zeta = \{T, \Theta\}$, and $\Theta(\bigcirc) = \bigcirc' : \mathbf{G}^{\mathbf{Y}} \times \mathbf{G}^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, then the corresponding *homomorphic computational system* (HCS) is given by $X_{\bigcirc}$, which is a CCS if $CR(T) > 1$.

Case 4. If m = 2, then let $A_1 = \mathbf{F}^{\mathbf{X}}$, $A_2 = (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, $B_1 = \mathbf{G}^{\mathbf{Y}}$, $B_2 = (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}$, and $\bigcirc = \textcircled{\scriptsize T}: \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$. Let $\zeta = \{T, U; \Theta\}$, where $U: (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}$ incorporates the transformation process inherent in T such that $\Theta(\bigcirc) = \bigcirc' : \mathbf{G}^{\mathbf{Y}} \times (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is an analogue of $\bigcirc$ over *range*(T). Given image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, if $\mathbf{a}_c = T(\mathbf{a})$ and $\mathbf{s} = U(\mathbf{t})$, then

$$\Theta(\bigcirc)[T(\mathbf{a}), U(\mathbf{t})] = T(\bigcirc(\mathbf{a}, \mathbf{t})) = T(\mathbf{a} \textcircled{\scriptsize T} \mathbf{t}). \tag{79}$$

This implies the existence of the structure

$$X_{\textcircled{\scriptsize T}} = (\mathcal{A}, \zeta, \mathcal{B}) = \left(\left\{\mathbf{F}^{\mathbf{X}}, (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}; \textcircled{\scriptsize T}\right\}, \{T, U; \Theta\}, \left\{\mathbf{G}^{\mathbf{Y}}, (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}; \bigcirc'\right\}\right), \tag{80}$$

which is a CCS if $CR(T) > 1$.

Case 5. If m = 2, then let $\mathcal{A} = \left\{\mathbf{F}^{\mathbf{X}}, (\mathbf{F}^{\mathbf{X}})^{\mathbf{V}}, \mathbf{F}^{\mathbf{V}}; \textcircled{\scriptsize T}\right\}$, $\mathcal{B} = \left\{\mathbf{G}^{\mathbf{Y}}, (\mathbf{G}^{\mathbf{Y}})^{\mathbf{W}}, \mathbf{G}^{\mathbf{W}}; \bigcirc'\right\}$, $\overline{T}: \mathbf{F}^{\mathbf{V}} \rightarrow \mathbf{G}^{\mathbf{W}}$, $U: (\mathbf{F}^{\mathbf{X}})^{\mathbf{V}} \rightarrow (\mathbf{G}^{\mathbf{Y}})^{\mathbf{W}}$, and $\zeta = \{T, \overline{T}, U; \Theta\}$, where $\Theta(\bigcirc) = \bigcirc'$ is an analogue over *range*(ζ) of $\textcircled{\scriptsize T}$. If T and $\overline{T}$ are compressive transforms, then setting $X_{\textcircled{\scriptsize T}} = \{\mathcal{A}, \zeta, \mathcal{B}\}$ yields the CCS for the generalized image-template product over heterogeneous domains.

3.2.2.17. Remark. Definition 3.2.2.14 and Observation 3.2.2 provide a unifying formalism that supports the derivation of analogues of nonrecursive image algebra operations over *range*(T). Note that Cases 4 and 5 of the preceding observation can describe operations between templates without loss of specificity, since $\mathbf{t}_{\mathbf{y}} \in \mathbf{F}^{\mathbf{X}}$ for all $\mathbf{y} \in \mathbf{X}$. Additionally, duals of $\bigcirc$ over *range*(ζ) are expressed symmetrically to Cases 1–5. For example, if Γ has a dual Γ' over *range*(ζ), then Case 2 would yield the dual CCS

$$\overline{X}_{\Gamma} = (\{\mathbf{F}^{\mathbf{X}}, \mathbf{F}; \Gamma\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \mathbf{F}; \Gamma'\}). \tag{81}$$

In passing, we note that compressive transforms are generally not based upon a scalar mapping of form $\mathsf{F} \rightarrow \mathsf{G}$, as are pointwise greylevel operations. Rather, image compression is generally achieved by mapping source image blocks to one or more exemplar values $g \in \mathsf{G}$. For example, g could be an exemplar that approximately characterizes an encoding block, as in VQ. As a result, the global reduce operations usually do not have numerous analogues, since a single value in G represents many values in F, but rarely one value. However, the block mapping $\mathsf{F}^{kl} \rightarrow \mathsf{G}$ that is often inherent in T facilitates the construction of a mapping $g: \mathsf{G} \rightarrow \mathsf{F}^{kl}$ from which a dual $\Gamma': \mathsf{G}^{\mathbf{Y}} \rightarrow \mathsf{F}$ can be derived via applying Γ to *range*(g). We thus have discovered few systems of form X_{Γ}, but have derived numerous CCS of form $\overline{X}_{\Gamma}$.

3.2.2.18. Observation. If all the elements of ζ except Θ (i.e., $\zeta_1, \zeta_2, \ldots, \zeta_n$) are *onto* mappings that have inverses denoted by $\xi = \{\zeta_1^{-1}, \zeta_2^{-1}, \ldots, \zeta_n^{-1}\}$, then ζ is an algebraic isomorphism. If $\bigcirc$ is a binary operation and there is only one operand set, then $X_{\bigcirc}$ can be recast as an *isomorphic computational system* (ICS), as follows:

$$V_{\bigcirc} = (\mathcal{A}, \zeta, \mathcal{B}, \xi) = \left(\{\mathsf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathsf{G}^{\mathbf{Y}}, \bigcirc'\}, T^{-1}\right). \tag{82}$$

It is well known that ICS are useful in cryptology, since isomorphisms are required for exact recovery of plaintext from ciphertext.

If any of the elements of ζ except Θ does not have an inverse, then we can construct ϵ-near approximations to the inverses of $\zeta_1, \zeta_2, \ldots, \zeta_n$ that are denoted by $\xi^* = \{\zeta_1^*, \zeta_2^*, \ldots, \zeta_n^*\}$. In such cases, we say that there exists an *approximate ICS* $V_{\bigcirc}^{\sim} = (\mathcal{A}, \zeta, \mathcal{B}, \xi^*)$.

We next overview applications of the preceding theory to high-level problems in computer science, in preparation for the more specific discussion of transformation and compressive processing given in Sections 3.3 and 3.4.

3.2.3. Relationship of Compressive Processing to Problems of Computer Science.

We previously mentioned that the derivation of specific solutions of the transform properties problem was central to the feasibility of compressive processing. The TPP provides a convenient, unifying formalism for expressing problems of computer science that pertain to imaging practice. In particular, portions of our research currently emphasize the semi-automatic derivation of partial functions that approximate forward and inverse transformations. For example, consider the problem of inverting an imaging system's point-spread function (PSF) that is approximately deduced from imagery. Inversion is but one step in deconvolving a PSF from camera imagery, in order to clarify details of interest. Similar inversion capabilities would be useful in cryptanalysis, where an adversary attempts to guess the key (or the mechanism of) an encryption transform.

In this section, we show that formal statements of the TPP and several related problems are useful for expressing homomorphic and isomorphic computational systems. Through such discussion, we hope to acquaint the reader with the larger, conceptual context of this study, prior to addressing applications of our theory in the remaining chapters. Eventually, the semi-automatic development of approximations to solutions of the TPP could facilitate further automation of burdensome computing tasks (e.g., derivation of compression and encryption algorithms) that are currently performed manually or semi-automatically. Thus, the sense and import of this section is not merely abstract, but has potential applications in computing practice that are being actively investigated in our ongoing research.

We begin by defining Kleene closure in terms of functional composition.

3.2.3.1. Observation. In programming language theory, it is customary to define Kleene closure in terms of concatenation. For example, given the set of symbols A, let L, L_1, and L_2 denote *strings* that are comprised of symbols in A. The *concatenation* of L_1 and L_2, denoted by L_1L_2, is equal to the set $\{xy\colon x \in L_1 \text{ and } y \in L_2\}$. Let us further define

the set $L^0 = \{\lambda\}$, where λ denotes the *empty symbol*, and $L^i = LL^{i-1}$, for $i \geq 1$. Defined in terms of the concatenation relation, the Kleene closure of L, denoted by L^*, and the *positive closure of* L, denoted by L^+, are given by

$$L^* = \bigcup_{i=0}^{\infty} L^i \quad \text{and} \quad L^+ = \bigcup_{i=1}^{\infty} L^i . \tag{83}$$

A corresponding definition is given for tuple construction, as follows.

3.2.3.2. Definition. Given a set of operators $\mathcal{O}$, let O, O_1, and O_2 denote *sequences of operators* that are comprised of operators in $\mathcal{O}$. The *construction* of O_1 and O_2, denoted by $O_1 \tau O_2$, is equal to the set of tuples $\{(\bigcirc, \Omega) : \bigcirc \in O_1 \text{ and } \Omega \in O_2\}$. Further define $O^0 = \Xi$, where Ξ denotes the *null operator* (which does nothing), and $O^i = O \tau O^{i-1}$, for $i \geq 1$. The Kleene closure of O under the construction relation, denoted by $O^{*\tau}$, is given by

$$O^{*\tau} = \bigcup_{i=0}^{\infty} O^i . \tag{84}$$

For purposes of brevity (and to avoid typographic confusion among the sequence O and the operators $\bigcirc$ and $\circ$), we hereafter denote Kleene closure under the construction relation as $\mathcal{O}^* \equiv O^{*\tau}$. The construction relation τ can be likewise applied to operands in $\mathcal{F}$ to yield the Kleene closure $\mathcal{F}^*$.

3.2.3.3. Example. Let $\mathcal{O} = \{+, \cdot\}$ and let Ξ denote the *null operator*. The elements of $\mathcal{O}^*$ are specified in canonical order as

$$\mathcal{O}^* = \{\Xi, +, \cdot, (\Xi, \Xi), (\Xi, +), (\Xi, \cdot), (\Xi, \Xi, \Xi), \ldots\} . \tag{85}$$

We thus have a convenient theoretical construct for enumerating sequences of operators. We call such a sequence an *algorithm*.

3.2.3.4. Notation. For purposes of discussion, we denote the set of *transforms* $\mathcal{T} \subset \mathcal{O}^*$.

3.2.3.5. Definition. The *data transformation problem* (DTP) can be described in simplified form by the mapping $M_{DTP} : (S' \cup \{\perp\})^S \times \mathcal{A} \rightarrow \mathcal{T}$, if the following conditions are satisfied:

(a) *domain*(M_{DTP}) consists of

(i) a set D of ordered tuples (f,f') that constitute a partial function, where $f \in S$, and $f' \in S'$, with $S, S' \in \mathcal{F}$ the domain and range spaces of a transform $T \in \mathcal{T}$, and $\perp$ denotes an unidentified entity;

(ii) an algebra $\mathcal{A} = (\mathcal{G}, \mathcal{U})$, where $\mathcal{G} \subset \mathcal{F}$ denotes a set of data structures and $\mathcal{U} \subset \mathcal{O}$ denotes a set of operators;

(b) *range*(M_{DTP}) contains the transform $T: S \rightarrow S'$; and

(c) $M_{DTP} : (D, (\mathcal{G}, \mathcal{U})) \rightarrow T$.

3.2.3.6. Remark. For purposes of consistency, we currently assume that M_{DTP} expresses T in terms of operators in $\mathcal{U}$ as well as operands (e.g., data structures) in $\mathcal{G}$.

Since each transform T can be written in various ways, the consideration of low-level formulations of M_{DTP} is presently tangential to our theoretical development. The reader should not confuse an abstract formulation of M_{DTP} with more concrete solutions to instances of the DTP. For example, we note that closed-form solutions to the DTP are not evident in the literature for sets S,S' of practical interest. However, Koza [58] has demonstrated methods of deriving exact or approximate solutions to restricted instances of the DTP over specific finite discrete sets S,S' via stochastic techniques. We next examine the problem of obtaining inverse transformations, which closely resembles the DTP.

3.2.3.7. Definition. The *inverse transformation problem* (ITP) can be described in simple form by the mapping $M_{ITP} : \mathcal{T} \times \mathcal{A} \rightarrow \mathcal{T}$, if the following conditions are satisfied:

(a) *domain*(M_{ITP}) consists of

(i) a transform $T: S \rightarrow S'$ in $\mathcal{T}$, where $S, S' \in \mathcal{F}$, the operands;

(ii) an algebra $\mathcal{A} = (\mathcal{G}, \mathcal{U})$, where $\mathcal{G} \subset \mathcal{F}$ denotes a set of data structures that instantiate S and S', and $\mathcal{U} \subset \mathcal{O}$ denotes a set of operators from which the transform in *range*(M_{ITP}) is comprised;

(b) The inverse transform $T^{-1}: S' \rightarrow S$ exists and is contained in *range*(M_{ITP}); and

(c) $M_{ITP} : (T, (\mathcal{G}, \mathcal{U})) \rightarrow T^{-1}$.

Solutions to instances of M_{ITP} have been employed in cryptanalysis, e.g., the determination of a monoalphabetic substitutional cipher using an input comprised of unique symbols. Recently, there was reported a successful attack upon rotor machines via genetic algorithms (GAs) using sample plaintext, ciphertext, and a knowledge of key size [34].

Instead of deriving the inverse transformation in closed form from the forward transform, it may be more efficient to approximate the inverse transform computationally. For example, the forward transform can be represented discretely as a partial function, to which an approximate inverse can be sought. Such derivation of approximate inverse transforms is expressed similarly to the ITP, as shown in the following definition.

3.2.3.8. Definition. An instance of the *approximate inverse transformation problem* (AIP) can be described in simple form by the mapping $M_{AIP} : (S' \cup \{\perp\})^S \times \mathcal{T} \times \mathbf{R} \times \mathcal{A} \rightarrow \mathcal{T}$, if the following conditions are satisfied:

(a) *domain*(M_{AIP}) consists of

(i) a set D of ordered tuples (f,f') that constitute a partial function, where $f \in S$, $f' \in S'$, and $S, S' \in \mathcal{F}$ are the domain and range spaces of a transform $T \in \mathcal{T}$, with $\perp$ denoting an unidentified entity;

(ii) a transform $T: S \rightarrow S'$ in $\mathcal{T}$;

(iii) an error bound $\epsilon \in \mathbb{R}$;

(iv) an algebra $\mathcal{A} = (\mathcal{G}, \mathcal{U})$, where $\mathcal{G} \subset \mathcal{F}$ denotes a set of data structures that instantiate S and S', and $\mathcal{U} \subset \mathcal{O}$ denotes a set of operators of which the transform in $range(M_{AIP})$ is comprised;

(b) $range(M_{AIP})$ contains an ϵ-near inverse approximation to T, denoted by $T^*: S' \rightarrow S$; and

(c) $M_{AIP} : (D, T, \epsilon, (\mathcal{G}, \mathcal{U})) \rightarrow T^*$.

3.2.3.9. Observation. Inherent in the preceding definition is the assumption that T^* can be constructed and applied to the input data in D within the error bound ϵ. In practice, such may not be the case. For example, one may have to select some $Q \subset S$ and $Q' \subset S'$ in order for $T^*: Q' \rightarrow Q$ to satisfy ϵ. In such cases, it is reasonable to assume that $range(T^*) \subset S$.

3.2.3.10. Example. Let a polynomial regression technique be applied to a dataset D. Assuming that the transform $T: S \rightarrow S'$ is also a polynomial, the coefficients of T could seed the regression that would determine a polynomial P which would approximate T. Then, one might invert P (where possible), thus obtaining a polynomial P^{-1} that could be employed to seed a polynomial regression over the set $D^* = \{(f', f) : (f, f') \in D\}$. Again applying regression to D^* to yield a polynomial T^*, analysis of residuals in the output of T^* could highlight errors that lie outside the interval $[-\epsilon, \epsilon]$. That is, T^* would exhibit errors within the bounds $\pm\epsilon$ only for some pairs $(f', f) \in D^*$. In such cases, we would rewrite an instance of Definition 3.2.3.8 to read $T^*: Q' \rightarrow Q$, where $Q \subset S$ and $Q' \subset S'$.

3.2.3.11. Remark. The preceding observation implies that formulations of M_{AIP} may be constrained by prespecification of the error bound ϵ in *domain*(M_{AIP}). This is a realistic constraint, since postspecification of ϵ in *range*(M_{AIP}) implies loss of precise control over the error bound, which precision could be required in engineering applications.

As a result of such considerations, we present Definition 3.2.3.8 as one of many instances of a more general problem. Additional issues of interest, e.g., whether M_{AIP} reduces to an instance of M_{ITP} when $\epsilon = 0$, will be considered in future research. We next summarize the central problem of this study, namely, that of determining properties of a given transform.

3.2.3.12. Definition. The *transform properties problem* (TPP) can be specified in terms of the mapping $M_{TPP} : (\mathcal{F} \times \mathcal{O}) \times T \times \mathcal{A} \rightarrow \mathcal{F} \times \mathcal{O}$, if the following conditions are satisfied:

(a) *domain*(M_{TPP}) consists of

(i) a tuple $(S, \bigcirc)$ comprised of a value set $S \in \mathcal{F}$ and an operation $\bigcirc \in \mathcal{O}$;

(ii) a homomorphism T in $\mathcal{T}$, from S to S';

(iii) an algebra $\mathcal{A} = (\mathcal{G}, \mathcal{U})$, where $\mathcal{G} \subset \mathcal{F}$ denotes a set of data structures and $\mathcal{U} \subset \mathcal{O}$ denotes a set of operators;

(b) *range*(M_{TPP}) contains a tuple $(S', \bigcirc')$, such that $S' = T(S)$ and $\bigcirc' \in \mathcal{O}^*$ is an operation on S'; and

(c) $M_{TPP} : ((S, \bigcirc), T, (\mathcal{G}, \mathcal{U})) \rightarrow (S', \bigcirc')$.

3.2.3.13. Observation. As mentioned previously, this study emphasizes the derivation of solutions to restricted instances of the TPP. For purposes of discussion, we define the term *feasible computation* to mean an algorithm that satisfies implementational constraints of time, space, and error for a certain operational scenario. In this study, we concentrate on designing feasible computations over the range space of various transforms. In Section 3.2.2, we defined a computation over *range*(T) in terms of computational systems whose derivation was based on an algebraic homomorphism. Given such definitions, we next

observe that HCS and ICS can be constructed from solutions to the TPP, ITP, and AIP. The following theorems illustrate this concept and have simple proofs that are presented in outline form.

3.2.4. Derivation of HCS and ICS via the DTP, ITP, and AIP

3.2.4.1. Theorem. If particular solutions to the DTP and TPP exist, a homomorphic computational system X_H (reference Observation 3.2.2) can be constructed from (a) a set $D = \{(f, f') : f \in S, f' \in S'\}$, which is a partial function that is an element of $(S' \cup \{\perp\})^S$, and (b) an operation $\bigcirc$ on S.

Proof. Assuming the existence of an algebra $\mathcal{A} = (\mathcal{G},\mathcal{U})$, where $\mathcal{G} \subset \mathcal{F}$ and $\mathcal{U} \subset \mathcal{O}$, we outline the proof as follows:

Step 1. Choose $\bigcirc$, an operation on S.

Step 2. Where possible, solve the data transformation problem as

$$T := M_{DTP}(D, (\mathcal{G},\mathcal{U})), \tag{86}$$

to obtain the transformation $T : S \rightarrow S'$.

Step 3. Where possible, solve the transform property problem as

$$(S', \bigcirc') := M_{TPP}((S, \bigcirc), T, (\mathcal{G},\mathcal{U})), \tag{87}$$

to obtain the tuple $(S', \bigcirc')$. By the definition of an HCS, T is a homomorphism.

Step 4. Formulate $X_H = (\{S, \bigcirc\}, \{T, \Theta\}, \{S', \bigcirc'\})$, where an analogous operation $\Theta(\bigcirc) = p_2[M_{TPP}((S, \bigcirc), T, (\mathcal{G},\mathcal{U}))]$. □

3.2.4.2. Theorem. If particular solutions to the inverse transformation problem exist, an isomorphic computational system V can be derived from a homomorphic computational system X_H.

Proof. Assuming the existence of an algebra $\mathcal{A} = (\mathcal{G},\mathcal{U})$, where $\mathcal{G} \subset \mathcal{F}$ and $\mathcal{U} \subset \mathcal{O}$, we outline the proof as follows:

Step 1. Choose an HCS $X_H = ((S,\bigcirc),T,(S',\bigcirc'))$, where the transformation $T: S \to S'$.

Step 2. Where possible, solve the inverse transformation problem as follows:

$$T^{-1} := M_{ITP}(T,(\mathcal{G},\mathcal{U})), \tag{88}$$

to obtain the inverse transformation $T^{-1}: S' \to S$.

Step 3. Formulate $V = (\{S,\bigcirc\},\{T,\Theta\},\{S',\bigcirc'\},T^{-1})$, where an analogous operation $\Theta(\bigcirc) = p_2[M_{TPP}((S,\bigcirc),T,(\mathcal{G},\mathcal{U}))]$. □

3.2.4.3. Theorem. If particular solutions to the approximate inverse transformation problem exist, an approximate isomorphic computational system X_{I^*} can be derived from a homomorphic computational system X_H.

Proof. Assume the existence of an algebra $\mathcal{A} = (\mathcal{G},\mathcal{U})$, where $\mathcal{G} \subset \mathcal{F}$ and $\mathcal{U} \subset \mathcal{O}$, together with the dataset $D = \{(f,f') : f \in S, f' \in S'\}$, which is a partial function that is an element of $(S' \cup \{\perp\})^S$ with $S,S' \in \mathcal{F}$. The proof is outlined as follows

Step 1. Choose an HCS $X_H = ((S,\bigcirc),T,(S',\bigcirc'))$, where the homomorphism $T: S \to S'$.

Step 2. Choose an error bound $\epsilon \in \mathbf{R}$ and, where possible, solve the AIP as

$$T^* := M_{AIP}(D,T,\epsilon,(\mathcal{G},\mathcal{U})), \tag{89}$$

to obtain an ϵ-near approximation $T^*: S' \to S$ to the inverse transformation $T^{-1}: S' \to S$.

Step 3. Formulate $V^* = (\{S,\bigcirc\},\{T,\Theta\},\{S',\bigcirc'\},T^*)$, where an analogous operation $\Theta(\bigcirc) = p_2[M_{TPP}((S,\bigcirc),T,(\mathcal{G},\mathcal{U}))]$. □

We next present a taxonomy of image transformations that facilitates derivation of analogous and dual operations throughout the remainder of this dissertation.

3.3. Taxonomy of Image Transformations.

We begin by deriving a simple notation for transform classification and exemplify the computation of a transform parameter (e.g., compression ratio) for each class.

3.3.1. Taxonomic Classes.

From Chapter 2, recall that an image is a mapping of form $\mathbf{F}^{\mathbf{X}} \equiv \mathbf{X} \rightarrow \mathbf{F}$ and image transforms map images to images. In this study, image transforms, denoted by $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, are classified via taxa that describe equality of domain or range spaces.

3.3.1.1. Assumption. Although the general form of $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is mathematically the same whether or not $\mathbf{X} = \mathbf{Y}$ or $\mathbf{F} = \mathbf{G}$, numerous transforms of practical interest exhibit different range and domain spaces. Thus, we assume (for implementational convenience) that $\mathbf{F}^{\mathbf{X}}$, $\mathbf{F}^{\mathbf{Y}}$, $\mathbf{G}^{\mathbf{X}}$, and $\mathbf{G}^{\mathbf{Y}}$ denote distinct spaces.

3.3.1.2. Definition. Under constraint of Assumption 3.3.1.1, encoding transforms are grouped as follows:

Class 1: $T_1: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$

Class 2: $T_2: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{X}}$

Class 3: $T_3: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$

Class 4: $T_4: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$

3.3.1.3. Example. Class 1 transforms are exemplified by the

(a) Linear inner-product transforms (e.g., Fourier, Walsh, Hadamard, and Cosine transformations);

(b) Unary arithmetic and logic functions;

(c) Delta modulation transform, derivative-based compressions, PCM, and certain instances of DPCM;

(d) Substitutional ciphers over $\mathbb{F}$; as well as

(e) Ciphers such as DES and RSA, if the keyspace is not included in *domain*(T).

3.3.1.4. Example. Class 2 transforms include

(a) Local averaging operations;

(b) Thresholding and bit-slicing operations; and

(c) Image labelling operations such as connected component labelling.

3.3.1.5. Example. Class 3 transforms are represented by

(a) Spatial warping transforms (i.e., the affine transform);

(b) Image minification by subsampling or magnification by pixel replication; and

(c) Spatial superresolution transformations, such as sub-pixel interpolation.

3.3.1.6. Example. Class 4 contains many interesting transformations currently in use, which include

(a) Runlength encoding and sparse matrix reduction techniques;

(b) Fractal-based encryption via iterated function systems;

(c) HVS-based blockwise transforms such as VPIC;

(d) Blockwise sine/cosine transforms such as DCT, JPEG, and MPEG;

(e) Blockwise statistical transformations, including BTC and VQ;

(f) Adaptive or recursive encoding schemes, such as wavelet transformation, adaptive vector quantization, as well as adaptive Lempel-Ziv encoding; and

(g) Higher-level processes, such as the transformation of image components into adjacency graphs.

The foregoing transform classes each have different attributes (e.g., compression ratio). For example, computational complexity, data security, and information loss are discussed in Chapters 4 and 5. To demonstrate the utility of our taxonomy, we analyze the compressive properties of transforms in Classes 1–4, which provides background for the development of Parts II and III.

3.3.2. Compression Ratio.

3.3.2.1. Lemma. The compression ratio of a Class 1 transform T_1 is unitary.

Proof. From Definitions 3.3.1.2 and 3.2.1.8, the Class 1 transform T_1: $\mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ exhibits the compression ratio

$$\begin{aligned} \mathrm{CR}(T_1) &= \frac{|domain(choice(domain(T_1)))| \cdot siz[range(choice(domain(T_1)))]}{|domain(choice(range(T_1)))| \cdot siz[range(choice(range(T_1)))]} \\ &= \frac{|\mathbf{X}| \cdot siz(\mathbf{F})}{|\mathbf{X}| \cdot siz(\mathbf{F})} = 1 . \end{aligned} \tag{90}$$

As a result, Class 1 transforms alone are not interesting for compression applications. □

The following lemmas are similarly proven.

3.3.2.2. Lemma. The compression ratio of Class 2 transforms is given by $\mathrm{CR}(T_2) = siz(\mathbf{F})/siz(\mathbf{G})$.

3.3.2.3. Lemma. The compression ratio of Class 3 transforms is given by $\mathrm{CR}(T_3) = |\mathbf{X}|/|\mathbf{Y}|$.

3.3.2.4. Lemma. The compression ratio of Class 4 transforms is given by

$$\mathrm{CR}(T_4) = \frac{|\mathbf{X}| \cdot siz(\mathbf{F})}{|\mathbf{Y}| \cdot siz(\mathbf{G})}, \tag{91}$$

which is the general case.

For purposes of clarity, we next present a notation for decomposing transform mappings, which helps us visualize the transform mappings implementationally.

3.3.3. Formulaic Granularity and Decomposition.

We begin by showing how a given transform T can be expressed at three levels of granularity (coarse, medium, and fine). The multigranular representation of T does not imply that each level of granularity has the same formulation. Rather, we decompose T to concisely acquaint the reader with T's structure and function. From such decompositions, one can further understand the format in which information is represented in *range*(T).

3.3.3.1. Definition. An image transform is described at *coarse granularity* as a mapping between image spaces of form $\mathbf{F}^{\mathbf{X}}$. Coarse granularity is denoted by presuperscript C. For example, we write sparse matrix reduction at coarse granularity as ${}^{\mathrm{C}}T_{\mathrm{SM}}$: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, which is a Class 4 transform.

3.3.3.2. Definition. A transform can be described at *medium granularity* via its data structures. Medium granularity is denoted by presuperscript M. For example, given $\mathbf{Y} \subset \mathbf{N}$ an instance of sparse-matrix reduction is denoted as ${}^{\mathrm{M}}T_{\mathrm{SM}}$: $\mathbf{F}^{\mathbf{X}} \rightarrow (\mathbf{X} \times \mathbf{F})^{\mathbf{Y}}$. That is, an $\mathbf{F}$-valued image on $\mathbf{X}$ is reduced to a list on $\mathbf{Y}$ of tuples of form $(\mathbf{x},\mathbf{f})$, where a non-negligible value $\mathrm{f} \in \mathbf{F}$ is grouped with its domain point $\mathbf{x} \in \mathbf{X}$.

3.3.3.3. Definition. The *fine granular* level of transform description can specify operations on one or more source data. Such operations may be represented as scalar (as opposed to vector-space) mappings. Fine granularity is denoted by presuperscript F. For example, given the preceding definition, we can write

$$^{F}T_{SM}\colon (\mathbf{x},\mathbf{a}(\mathbf{x}))\mapsto (h(\mathbf{x},\mathbf{a}(\mathbf{x})),(\mathbf{x},\mathbf{a}(\mathbf{x}))) \tag{92}$$

where h denotes an indexing function that is described in greater detail in Chapter 9.

3.3.3.4. Notation. We denote the *formulaic decomposition* of a transform T from coarse- to fine-granular form as $^{C}T \triangleq {}^{M}T \triangleq {}^{F}T$.

Two examples of this technique follow, which are applied to runlength encoding (RLE).

3.3.3.5. Example. Let the Class 4 mapping $T_4\colon \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ describe runlength encoding (RLE). The RLE transform T_{RLE} maps a Boolean image on $\mathbf{X}$ to an integer-valued image on $\mathbf{Y}$, where $\mathbf{Y} \subset \mathbf{N}$ is usual.

Let n denote the maximum source runlength in *domain*(T_{RLE}), and let a run (i.e., a contiguous region) of $k \leq n$ zero-valued (unitarily-valued) pixels be mapped to the integer $-k$ ($+k$). If $k > n$, then we partition the run into k/n runs of length n, and a residual run of length k mod n pixels. For example, if *range*(*choice*(*domain*(T_{RLE}))) = $\mathbf{Z}_{256}$, and we want the encoding alphabet (which includes a marker symbol) to be both positively and negatively valued, then $n = \lfloor (256-1)/2 \rfloor = 127$ pixels. Since we will be accumulating the detected runs using a counter, we let $\mathbf{Y} = \mathbf{N}$. Denoting $\mathbf{Z}_l^{\pm} = \{i : i \in [-(l-1), l-1] \subset \mathbf{Z}\}$, we denote *range*(*choice*(*range*(T_{RLE}))) as $\mathbf{Z}_{n+1}^{\pm}$. Thus, we write $^{M}T_{\text{RLE}}\colon \mathbf{Z}_2^{\mathbf{X}} \rightarrow (\mathbf{Z}_{n+1}^{\pm})^{\mathbf{N}}$.

Per the preceding discussion (and as shown in Chapter 9), $^{F}T_{\text{RLE}}\colon \{f\}^{k} \mapsto k \cdot (2f-1)$, where $\{f\}^{k}$ denotes a zero- or unitarily-valued run of k pixels, and $k \leq n$.

3.3.3.6. Example. The Class 4 transforms, which can be complex mathematically as well as notationally, can be further classified as follows:

(i) Class 4–A contains *fixed-blocksize* substitutions, denoted by T_{4f}, where

$$ {}^{M}T_{4f}: \left(\mathbf{F}^{k}\right)^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}} \quad \triangleq \quad {}^{F}T_{4f}: (f_1, f_2, \ldots, f_k) \mapsto g, f_1, f_2, \ldots, f_k \in \mathbf{F}, g \in \mathbf{G}. \quad (93) $$

Thus, the encoding block size k is invariant to the points $\mathbf{x} \in \mathbf{X}$ or $\mathbf{y} \in \mathbf{Y}$.

(ii) Class 4–B contains *variable-block* substitutions, denoted by T_{4v}, where the medium-granular transform appears as in the previous equation, but the fine-granular transform exhibits space-variant encoding blocksize, i.e.,

$$ {}^{F}T_{4v}: (\mathbf{y}, (f_1, f_2, \ldots, f_{k(\mathbf{y})})) \mapsto (\mathbf{y}, g), f_1, f_2, \ldots, f_{k(\mathbf{y})} \in \mathbf{F}, g \in \mathbf{G}, \mathbf{y} \in \mathbf{Y}. \quad (94) $$

It is easily verified that Class 4–B portrays substitutions whose blocksize varies with $\mathbf{x} \in \mathbf{X}$, for example, partitioning systems customarily employed in fractal-based compression. Given $\mathbf{Y}$, the indexing function $h: \mathbf{X} \rightarrow \mathbf{Y}$, and an input image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ in *domain*(T_4), Equation 94 can be rewritten to portray a *value-dependent* blocksize. In such cases, T_4 would portray an encoding with a blocksize that is dependent upon the values of $\mathbf{a}$, since $\mathbf{a}$ is a function from $\mathbf{X}$ to $\mathbf{F}$.

The medium- and fine-granular transform descriptions are especially useful when analyzing a mapping's computational complexity and data security, as shown in Chapter 4. Chapter 5, which pertains to error analysis, employs the fine-granular formulation as an initial step in characterizing error propagation through various transform stages. The derivations and implementational analyses of Chapters 6–9 are also based partially upon multi-granular decomposition. Thus, our notation is used not only for classification, but facilitates theoretical derivation, analysis, and algorithmic design.

We next discuss high-level methods for the derivation of operations that compute over the range spaces of Class 1–4 transforms.

3.4. Class-Specific Derivational Techniques.

We have thus far presented theory that is basic to the analysis of compressive and encryptive transformations, and have proposed a taxonomy by which numerous image transforms may be classified. In this section, we describe high-level techniques for analyzing transforms in taxonomic Classes 1–4 and for determining operations over the range spaces of such transforms.

We begin by summarizing a few simple algebraic methods that are useful analytically (Section 3.4.1), then progress to high-level derivations of pointwise operations (Section 3.4.2), global reduce, and image-template operations (Sections 3.4.3 and 3.4.4, respectively). Given such basic theory, we present more detailed descriptions of high-level methods for deriving analogues and duals of operations over imagery transformed by Class 1–4 mappings (Sections 3.4.5 through 3.4.8).

3.4.1. Basic Algebraic Methods.

3.4.1.1. Assumption. Hereafter assume that $f, f^{-1}, g, g^{-1}, h \in S$, where f^{-1} and g^{-1} denote the inverses of f and g. Additionally, we assume that the operations $\bigcirc, \gamma : S \times S \rightarrow S$, $\bigcirc', \gamma' : S' \times S' \rightarrow S'$, the transform $T : S \rightarrow S'$ is a homomorphism, and e denotes the identity of S with respect to $\bigcirc$.

3.4.1.2. Observation. Let Assumption 3.4.1.1 hold. One can employ the following simple techniques for the reduction of algebraic expressions:

(a) If $\bigcirc$ is commutative, then one can transpose the arguments of $\bigcirc$ as

$$f \gamma (g \bigcirc h) = f \gamma (h \bigcirc g), \tag{95}$$

(b) If $\bigcirc$ is associative, then one can regroup terms as

$$(f \bigcirc g) \bigcirc h = f \bigcirc (g \bigcirc h), \tag{96}$$

(c) If γ is associative with respect to $\bigcirc$, then one can regroup terms to yield

$$(f \bigcirc g)\gamma h = f \bigcirc (g \gamma h), \tag{97}$$

(d) If $\bigcirc$ is associative and g has an inverse $g^{-1} \in S$, then one cancels terms as

$$(f \bigcirc g) \bigcirc g^{-1} = f \bigcirc (g \bigcirc g^{-1}) = f \bigcirc e = f, \tag{98}$$

(e) If $\bigcirc$ is associative and has an inverse $\bigcirc^{-1}$ that is associative with respect to $\bigcirc$, then one can eliminate terms, i.e.,

$$(f \bigcirc g) \bigcirc^{-1} g = f \bigcirc (g \bigcirc^{-1} g) = f \bigcirc e = f, \tag{99}$$

3.4.1.3. Example. Let $g = f^{-1} \gamma h$. Via substitution, $f \bigcirc g$ becomes $f \bigcirc (f^{-1} \gamma h)$. If γ is associative with respect to $\bigcirc$, then we have

$$f \bigcirc (f^{-1} \gamma h) = (f \bigcirc f^{-1}) \gamma h = e \gamma h = h \gamma e = h. \tag{100}$$

3.4.2. Pointwise Operations.

We first examine the simple case of image transforms that are bijections, then continue with spatial transforms, and non-invertible (lossy) m-ary mappings. We conclude our development with a discussion of fixed- and variable-blocksize encodings, which comprise the majority of compression transforms in common use.

3.4.2.1. Assumption. Let $V_{\bigcirc} = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{F}^{\mathbf{X}}, \bigcirc'\}, T^{-1})$ denote an ICS such that

(i) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is implemented in terms of a bijection $g: \mathbf{F} \rightarrow \mathbf{F}$, and

(ii) $\Theta(\bigcirc)(T(\mathbf{a})) = T(\bigcirc(\mathbf{a})), \forall \mathbf{a} \in \mathbf{F}^{\mathbf{X}}$.

3.4.2.2. Technique. If Assumption 3.4.2.1 holds, then for finite $\mathbf{F}$ and $\mathbf{X}$, we observe that Θ and hence, $\bigcirc' = \Theta(\bigcirc)$, can be implemented in terms of a discrete data structure such as a lookup table, subject to space constraints dictated by $|\mathbf{F}|$. Alternatively, if T is expressed symbolically (i.e., an analytic function) and we wish to express $\bigcirc'$ analytically, then we solve 3.4.2.1(ii) to obtain $\bigcirc'$ under constraints of tractability. The following example is illustrative.

3.4.2.3. Example. Let Assumption 3.4.2.1 hold, and let $T = \log$, which implies that $\mathbf{F} = \mathbf{R}^+$. If $\bigcirc = \cdot$, then one can solve the following equation analytically

$$\bigcirc'(\log f, \log g) = \bigcirc(f, g) = f \cdot g, \forall f, g \in \mathbf{R}^+ \tag{101}$$

to obtain $\bigcirc' = +$.

In order to better understand our treatment of neighborhood-based mappings, we next consider the simplistic case of an m-ary partial function that is a bijection over the domain of the image operation $\bigcirc$.

3.4.2.4. Assumption. Let $V_{\bigcirc} = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{F}^{\mathbf{X}}, \bigcirc'\}, T^{-1})$ denote an ICS such that

(i) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is based on a pointwise m-ary function $g: \mathbf{F}^{\mathbf{m}} \rightarrow \mathbf{F}$,

(ii) $T^{-1}: \mathbf{F} \rightarrow \mathbf{F}^{\mathbf{m}}$ exists only for those $|\mathbf{F}|$ neighborhood configurations in *domain*(T), and

(iii) $\Theta(\bigcirc)(T(\mathbf{a})) = T(\bigcirc(\mathbf{a}))$ for those $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ for which condition (ii) holds.

3.4.2.5. Observation. Note that there are $|\mathbf{F}|^{\mathbf{m}}$ possible neighborhood configurations in $\mathbf{F}^{\mathbf{m}}$, but only $|\mathbf{F}|$ mappings possible under constraint of the bijection that constitutes the partial function T. That is, if T is an *onto* mapping, then the inverse $T^{-1}: \mathbf{F} \rightarrow \mathbf{F}^{\mathbf{m}}$, if it exists, would be an *into* mapping.

3.4.2.6. Technique. The equation in Assumption 3.4.2.4(iii) can be solved for $\bigcirc'$, subject to the given constraints, after the method of Section 3.4.2.3. However care must be takin in qualifying constraints under which the solution holds. The following example is illustrative.

3.4.2.7. Example. Let Assumption 3.4.2.4 hold, and let $\mathbf{F} = \mathbf{Z}_9$ with $\mathbf{X}$ a 2×4–pixel array. Let the source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ shown in Figure 2 be transformed by $T: \mathbf{F}^4 \rightarrow \mathbf{F}$ to yield $\mathbf{a}_c \in \mathbf{F}^{\mathbf{X}}$, as shown in Figure 2b. Define T as follows, using the form $\mathbf{F}^{2\times2} \rightarrow \mathbf{F}^{2\times2}$ as applicable, for purposes of brevity:

(i) $T: \begin{pmatrix} f & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & f \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ f & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & f \end{pmatrix} \vdash f \in \mathbf{F}$, where f is a nonzero value in the 2×2–pixel array that is input to T;

(ii) $T: \begin{pmatrix} 0 & f \\ 0 & g \end{pmatrix} \vdash \begin{pmatrix} 0 & f \\ 0 & g \end{pmatrix}$ and $T: \begin{pmatrix} f & 0 \\ g & 0 \end{pmatrix} \vdash \begin{pmatrix} f & 0 \\ g & 0 \end{pmatrix}$, where nonzero $f, g \in \mathbf{F}$;

(iii) $T: \begin{pmatrix} e & f \\ g & h \end{pmatrix} \vdash \begin{pmatrix} e & h \\ g & f \end{pmatrix}$, where nonzero $e, f, g, h \in \mathbf{F}$; and

(iv) T is undefined elsewhere, for purposes of simplicity.

$$\begin{matrix} 1 & 2 & 7 & 5 \\ 3 & 4 & 8 & 6 \end{matrix} \qquad \begin{matrix} 1 & 4 & 8 & 5 \\ 3 & 2 & 7 & 6 \end{matrix} \tag{102}$$

(a) (b)

Figure 2. (a) Example source image $\mathbf{a}$ that is transformed by T to yield (b) image $\mathbf{a}_c$. Note reversal of the middle 4×4-pixel neighborhood only.

Let $\bigcirc: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ be given by $\bigcirc(\mathbf{a}) = (\mathbf{a} - 1)$ *mod* 9. For example, $\bigcirc(8) = 1$, $\bigcirc(1) = 0$, etc. For the formulations of T, $\bigcirc$, and $\mathbf{a}$ given in Figure 2 and items (i)-(iv), it is easily verified by inspection of Figure 2 that $\bigcirc$ preserves the ordering of pixel values in $\mathbf{a}$ as constrained by the ordering of $\mathbf{X}$. That is, $\bigcirc$ does not transpose values in $\mathbf{a}$. As a result, it is sufficient to state that $\bigcirc' = \bigcirc$ in this case. However, if $\bigcirc(\mathbf{a}) = (\mathbf{a} - 6)$ *mod* 9, then the right-hand two columns of pixels in $\mathbf{a}_c$ will be $\begin{pmatrix} 2 & 8 \\ 1 & 0 \end{pmatrix}$ instead of $\begin{pmatrix} 8 & 5 \\ 7 & 6 \end{pmatrix}$, and the constituent map $T: \begin{pmatrix} e & f \\ g & h \end{pmatrix} \vdash \begin{pmatrix} e & h \\ g & f \end{pmatrix}$ can no longer be inverted to produce $\mathbf{a}$ from $\mathbf{a}_c$ under constraint of the existing definition of T. In such cases, T^{-1} does not exist.

3.4.2.8. Remark. It is this difficulty with inverting constituent maps of T that renders neighborhood-based bijections difficult (and generally impractical) for image compression. In practice, a collection of neighborhoods (also called *vectors*) is often represented in terms of an *exemplar* that portrays a group of vectors within a given error criterion. Note that the preceding example shows why the determination of analogues to lossless (i.e., bijective) neighborhood transforms that are partial functions is dependent on the formulation of T, $\bigcirc$, and $\mathbf{a}$.

3.4.2.9. Observation. Note that the preceding example specifies T as a space-variant spatial transform and $\bigcirc$ as a grey-level transform. There, the processing sequence $\bigcirc'(T(\mathbf{a}))$ would yield a greylevel transform ($\bigcirc'$) of the neighborhood-based spatial transform (T). This is similar in concept to the Data Encryption Standard, where greylevel and spatial transformations are alternated as application of the XOR and S-box sub-stages of each DES transform stage. However, we have noted that in the preceding section that, unless T and $\bigcirc$ are chosen carefully in relation to $\mathbf{a}$, the transformed image $\mathbf{a}_c$ could have undefined values. Hence, T^{-1} or $\bigcirc'$ might not exist, or $T^{-1}(\bigcirc'(T(\mathbf{a})))$ might be undefined. Noninvertibility would render T useless for cryptographic applications, where isomorphisms are required.

However, we have found that this observation can be exploited to form the basis for a neighborhood-based transposition similar to that shown in Section 3.4.2.6(iii). By choosing T and $\bigcirc$ carefully, one can construct an encryption $T(\bigcirc(\mathbf{a}))$ that reliably encrypts prespecified sequences of symbols in the plaintext $\mathbf{a}$ but discards (or irreversibly transforms) other sequences. Here, we say that T *irreversibly* transforms a given plaintext $\mathbf{a}$ because the inverse transform $T^{-1}(T(\mathbf{a}))$ could exist outside the mappings that comprise T. Thus, a highly selective DES-like transform could be devised that would obliterate certain words or transform them into undefined values (also called *noise values*).

Alternatively, consider that the preceding concept can be applied inversely to the *decryption* process, since Assumption 3.4.2.4 defines V to be an ICS. In other words, instead

of obliterating certain symbolic sequences, the partial function T and the operation $\bigcirc$ could accept ciphertext and obliterate partial decryptions that did not meet prespecified constraints. Although we have not fully explored the potential of such operations, they appear to have greater utility in cryptanalysis than in cryptography. This is likely due to the fact that in cryptanalysis, one searches for a given set of plaintext sequences while attempting to process (decrypt) the ciphertext. Although the ciphertext sequences may be accurately surmised, the corresponding plaintext is usually guessed in a sketchy fashion only. Since this study does not include the derivation of novel cryptologic transformations, we defer further discussion of this technique to future research.

We next consider the case of spatial warping transforms (Class 3 of our taxonomy).

3.4.2.10. Assumption. Let $V_{\bigcirc} = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{F}^{\mathbf{X}}, \bigcirc'\}, T^{-1})$ denote an ICS such that

(i) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ is based on the spatial transform $g: \mathbf{Y} \rightarrow \mathbf{X}$ such that $T(\mathbf{a}) = \mathbf{a} \circ g, \forall \mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, where $\mathbf{a} \circ g \equiv \{(\mathbf{y}, \mathbf{a}(g(\mathbf{y}))) : \mathbf{y} \in \mathbf{Y}\}$.

(ii) $T^{-1}: \mathbf{F}^{\mathbf{Y}} \rightarrow \mathbf{F}^{\mathbf{X}}$ exists and is based upon the spatial transform $g^{-1}: \mathbf{X} \rightarrow \mathbf{Y}$ such that $\mathbf{a} = T(\mathbf{a}) \circ g^{-1}, \forall \mathbf{a} \in \mathbf{F}^{\mathbf{X}}$;

(iii) $\Theta(\bigcirc)(T(\mathbf{a})) = T(\bigcirc(\mathbf{a}))$.

3.4.2.11. Theorem. If Assumption 3.4.2.10 holds, then $\bigcirc'$, the analogue of $\bigcirc$ over *range*(T), is given by

$$\bigcirc' = \bigcirc \circ g. \tag{103}$$

Proof. Assuming that the givens hold, if g has an inverse $g^{-1}: \mathbf{X} \to \mathbf{Y}$, then $T^{-1}: \mathbf{F}^{\mathbf{Y}} \to \mathbf{F}^{\mathbf{X}}$ exists, such that

$$\mathbf{a} = T^{-1}(\mathbf{a}_c) = \mathbf{a}_c \circ g^{-1} = \mathbf{a} \circ g \circ g^{-1}. \tag{104}$$

Given a unary image operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ (chosen for purposes of simplicity), the analogous operation over *range*(T), denoted by $\bigcirc': \mathbf{F}^{\mathbf{Y}} \to \mathbf{F}^{\mathbf{Y}}$, can be derived from the preceding equation, as

$$\begin{aligned} \bigcirc'(\mathbf{a}_c) &= T_3(\bigcirc(T_3^{-1}(\mathbf{a}_c))) \\ &= (\bigcirc(\mathbf{a}_c \circ g^{-1})) \circ g \\ &= \bigcirc(\mathbf{a}) \circ g = (\bigcirc \circ g)(\mathbf{a}), \end{aligned} \tag{105}$$

since $\mathbf{a}_c = \mathbf{a} \circ g$ and composition is associative. □

3.4.2.12. Remark. The preceding theorem holds for spatial warping transforms g that have an inverse g^{-1}. However, Equation 105 also shows us that g^{-1} is not required in order to derive $\bigcirc'$ from $\bigcirc$. Since the spatial warping transforms constitute an important class of image processing transformations that includes the well-known affine transform, this is a useful result. Additionally, since the affine transform forms a basis for fractal-like encoding of images (a Class 4 transform), this result is pertinent to imaging practice.

3.4.2.13. Observation. The spatial warping transforms present a deceptively easy example of deriving analogues for Class 3 transforms. Consider the more difficult case, where T is as defined in Assumption 3.4.2.10 but $\bigcirc$ is also a spatial transform. In particular, let $\bigcirc(\mathbf{a}) = \mathbf{a} \circ e$, where $e: \mathbf{Y} \to \mathbf{X}$ is not necessarily the inverse of g and may perturb *domain*($\mathbf{a}$) nonlinearly or noninvertibly. Since composition is not commutative, the commutativity diagram of Figure 1, which illustrates $\bigcirc'(T(\mathbf{a})) = T(\bigcirc(\mathbf{a}))$, may not hold since $\bigcirc'(\mathbf{a} \circ g) \neq (\mathbf{a} \circ e) \circ g$ is possible for some e. That is, the equality $\bigcirc'(\mathbf{a}_c) = \bigcirc'(\mathbf{a} \circ g) = (\bigcirc \circ g)(\mathbf{a})$ may not hold because composition is not commutative. Thus, only certain types of spatial transforms are admitted by Theorem 3.4.2.11, namely, those that are commutative and invertible, as stated in the following theorem.

3.4.2.14. Theorem. Let Assumption 3.4.2.10 hold and let $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and $e\colon \mathbf{Y} \rightarrow \mathbf{X}$ be a spatial warping transform such that $\bigcirc(\mathbf{a}) = \mathbf{a} \circ e$. If e commutes with T's basis mapping g and both are invertible, then $\bigcirc$ has an analogue $\bigcirc' = \bigcirc \circ g$.

Proof. The proof follows directly from the proof of Theorem 3.4.2.11 and the preceding discussion. □

3.4.2.15. Remark. The non-commutativity that prevents Theorem 3.4.2.11 from holding for all (discrete as well as continuous) domains $\mathbf{X}$ is exemplified by the case of an affine transform g that is first applied to *discrete* imagery which is then transformed by $\bigcirc(\mathbf{a}) = \mathbf{a} \circ e$. In such cases, quantization and interpolation error dictates that $g \circ e \neq e \circ g$, although g and e may be linear and invertible within an prespecified error constraint ϵ. Thus, we prefer to depict discrete spatial transforms that involve interpolation and quantization as being lossy and having ϵ-near approximations to their inverses.

The preceding discussion of a lossy transform leads naturally to a discussion of lossy neighborhood mappings. Such mappings can be found in Classes 1, 3, and 4 of our taxonomy, but primarily comprise the majority of useful transforms in Class 4.

3.4.2.16. Definition. A *lossy neighborhood transform* $T_{\mathrm{LN}}\colon \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is a noninvertible, m-ary *onto* mapping , where $\mathbf{Y} = \mathbf{X}$ and $\mathbf{G} = \mathbf{F}$ are possible. T_{LN} is expressed at medium granularity as ${}^{\mathrm{M}}T_{\mathrm{LN}}\colon \mathbf{F}^{\mathbf{m}} \rightarrow \mathbf{G}$.

3.4.2.17. Example. A local averaging transform that incorporates thresholding of the averaged result is an example of a lossy neighborhood transform. Other common examples include median filtering and neighborhood-based contrast enhancement. In such cases, the source image cannot be exactly retrieved from the compressed image that has been operated upon by an analogous operation.

3.4.2.18. Observation. Since T_{LN} is an *onto* mapping but is many-to-one, it is not invertible. As a result, we cannot construct an ICS based on T_{LN}, and we must instead use an approximate ICS based on an ϵ-near approximation to the inverse of T_{LN}, as follows.

3.4.2.19. Assumption. Let the ϵ-near approximate ICS

$$V_{\bigcirc}^{*} = (\{\mathsf{F}^{\mathbf{X}}, \bigcirc\}, \{T_{\mathrm{LN}}, \Theta\}, \{\mathsf{G}^{\mathbf{Y}}, \bigcirc'\}, T_{\mathrm{LN}}^{*}) \tag{106}$$

such that T_{LN} conforms to Definition 3.4.2.16.

3.4.2.20. Technique. When deriving $\bigcirc'$ over *range*(T_{LN}), we primarily consider the following cases:

(i) Linear or nonlinear operations may comprise T_{LN}, such that at a given encoding block's target point $\mathbf{x} \in \mathbf{X}$, the transform output approximates the source image value $\mathbf{a}(\mathbf{x})$.

(ii) T_{LN} may be primarily comprised of a spatial transform $g\colon \mathbf{Y} \rightarrow \mathbf{X}$, with greylevel manipulations that only slightly perturb the transform output values with respect to the corresponding source image values. For example, consider image rotation with interpolation. In such cases, the transformed value $\mathbf{a}_{\mathrm{c}}(\mathbf{y})$, where $\mathbf{y} \in \mathbf{Y}$, closely approximates the source image value $\mathbf{a}(g(\mathbf{y}))$.

(iii) Occasionally, a subset of *range*$(\mathbf{a})$ is exactly represented in $\mathbf{a}_{\mathrm{c}} = T_{\mathrm{LN}}(\mathbf{a})$. For example, a constant-valued encoding block may be encountered by a compression transform that encodes blocks according to their mean values (e.g., BTC or VPIC).

If Case (i) or (ii) holds, then it is possible that the analogue $\bigcirc'$ of an image operation $\bigcirc$ may be approximated by $\bigcirc$ due to the approximation of values in *domain*(T_{LN}) by corresponding values in *range*(T_{LN}). In case (iii) the equality $\bigcirc' = \bigcirc$ holds only for neighborhoods in $\mathbf{a}$ and $\mathbf{a}_c$ that have values such that $\mathbf{a}|_{h^*(\mathbf{y})}$ can be retrieved from $\mathbf{a}_c(\mathbf{y})$, where $\mathbf{y} \in \mathbf{Y}$.

From such simple cases, we progress to the more realistic case of neighborhoods that are represented by an exemplar, under constraint of a prespecified error criterion.

3.4.2.21. Technique. Let a neighborhood transform $T_{\mathrm{LN}}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ be expressed at medium granularity as ${}^{\mathrm{M}}T_{\mathrm{LN}}: \mathbf{F}^{\mathbf{m}} \rightarrow \mathbf{G}$, and let T_{LN} generate a *codebook* $\mathbf{c} \in (\mathbf{F}^{\mathbf{m}})^{\mathbf{G}}$. Given a set B of encoding blocks in $\mathbf{F}^{\mathbf{m}}$, the codebook is constructed by a clustering algorithm that groups B into sets B(g), for each $\mathrm{g} \in \mathbf{G}$ for which an exemplar $\mathbf{c}(\mathrm{g})$ is to be specified. The extent of this process can be constrained by a prespecified limit ϵ on the error with which vectors in B(g) are represented by $\mathbf{c}(\mathrm{g})$. An additional constraint is the codebook size $|\mathbf{G}|$. For example, ϵ could be an MSE criterion, such that

$$\| \mathbf{b} - \mathbf{c}(\mathrm{g}) \| < \epsilon, \forall \mathbf{b} \in \mathrm{B}(\mathrm{g}). \tag{107}$$

In practice, T is often called a *vector quantization* transform. Since $\mathbf{c}(\mathrm{g})$ represents encoding blocks in B(g) within an error ϵ, we say that T_{LN} has an ϵ–near approximation to its inverse, denoted at medium granularity by ${}^{\mathrm{M}}T^{*}_{\mathrm{LN}}: \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{m}}$. As a result, there are two strategies that can be employed in processing an image $\mathbf{a}$ compressed by T_{LN} to yield $\mathbf{a}_c$:

Method 1. Decode the $\mathbf{y}$-th block in $\mathbf{a}_c$ by applying T^{*}_{LN} to the block index $\mathrm{g} \in \mathbf{G}$ to yield the $\mathbf{y}$-th source block $\mathbf{b}$. Use customary tree-based search strategies to find the exemplar $\mathbf{c}(\mathrm{h})$ that best represents $\mathbf{b}$. Encode the processed image $\hat{\mathbf{a}}_c = \bigcirc'(\mathbf{a}_c)$ as $\mathbf{a}_c(\mathbf{y}) = \mathrm{h}$.

Method 2. Alternatively, one can apply $\bigcirc$ to each codebook exemplar in $\mathbf{c}$ to yield a new codebook $\mathbf{d}$. Then, $\mathbf{a}_c$ references $\mathbf{d}$ instead of $\mathbf{c}$.

3.4.2:22. Observation. With respect to the preceding technique, note the following salient advantages and disadvantages:

Method 1. **Advantage:** The codebook is not modified, and thus retains the representation of the training set upon which T_{LN}'s design was based.

Disadvantage: The transformed image must be inverse-transformed blockwise prior to processing by an analogous or dual operation, which contradicts the key premise of compressive processing, namely, the processing of imagery in compressed form.

Disadvantage: Since $|\mathbf{Y}| < |\mathbf{X}|/\mathrm{m}$ is typical for compressive applications, a given exemplar is likely to be processed by $\bigcirc'$ more than once, which is inefficient.

Disadvantage: The block configuration represented by $\bigcirc(\mathbf{b})$ may not be represented by any exemplar in $\mathbf{c}$ within the error limit ϵ upon which the design and construction of $\mathbf{c}$ was based.

Method 2. **Advantage:** Each exemplar is processed only once, requiring $\mathbf{O}(||\mathbf{Y}|)$ work.

Advantage: The image need not be decompressed. This maintains such data security as may be inherent in the compressed format, since $\bigcirc'$ does not require an association between *range*($\mathbf{a}_c$) and *domain*($\mathbf{c}$). In fact, $\bigcirc'$ never sees the placement of source image values in $\mathbf{a}_c$, since $\bigcirc'$ inputs values in *range*($\mathbf{a}_c$) only as indices of $\mathbf{c}$.

Advantage: The new codebook represents blocks produced by $\bigcirc'(T^*_{\mathrm{LN}}(\mathbf{a}_c))$, instead of approximating the output of $\bigcirc'$ with an existing exemplar.

Disadvantage: A new codebook is produced, whose statistics may vary from those of $\mathbf{c}$, and which may not represent statistics of the training set from which $\mathbf{a}$ was abstracted.

3.4.2.23. Remark. The advantages of Method 3.4.2.21(1) outweigh those of Technique 3.4.2.21(2), with the exception that the codebook statistics can be altered by Method 2. One could argue that a solution to this problem involves unioning the codebooks $\mathbf{c}$ and $\mathbf{d}$ within a given error constraint on exemplar similarity. However, codebook size is thus increased, which is clearly disadvantageous for compressive processing, when the net efficiency $\eta = \mathbf{O}(\mathrm{CR_d})$. This problem is further addressed in Chapter 9, when we discuss VQ-based processing, together with the associated codebook representation error and resultant reconstruction error that appears after a compressive operation's output is decompressed. This is an important implementational issue, due to the accrued error that results from cascaded compressive operations.

We next consider problems of coordinate-set manipulation that occur during the processing of block-encoded imagery using analogues or duals of binary pointwise operations.

3.4.2.24. Observation. Let Assumption 3.4.2.19 hold, and let the lossy neighborhood transform T_{LN} tesselate two images $\mathbf{a}, \mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ thereby producing the corresponding compressed images $\mathbf{a_c}, \mathbf{b_c} \in \mathbf{G}^{\mathbf{Y}}$. For each $\mathbf{y} \in \mathbf{Y}$, $\mathbf{a_c}(\mathbf{y})$ references a block in $\mathbf{a}$ (and likewise for $\mathbf{b_c}$ and $\mathbf{b}$). If T_{LN} tessellates $\mathbf{a}$ and $\mathbf{b}$ identically, then it is possible that $\bigcirc'$ can process over $\mathbf{a_c}$ and $\mathbf{b_c}$ in $\mathbf{O}(|\mathbf{Y}|)$ time since the $\mathbf{y}$-th blocks of $\mathbf{a_c}$ and $\mathbf{b_c}$ can be input to $\bigcirc'$. Thus, only $|\mathbf{Y}|$ invocations of $\bigcirc'$ are required.

However, this condition occurs only when T_{LN} employs an indexing function $h : \mathbf{X} \to \mathbf{Y}$ that identically tessellates the domains of $\mathbf{a}$ and $\mathbf{b}$. If h produces encoding block domains that are space-invariant, we call this *fixed-block encoding.* If *domain*($\mathbf{a}$) is tessellated isomorphically to *domain*($\mathbf{b}$), we have *isomorphic encoding*, which is usually employed in conjunction with fixed-block encoding.

3.4.2.25. Observation. In contrast to the preceding case, if source images $\mathbf{a}, \mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ are tessellated differently, then it is implied that the indexing functions $h\colon \mathbf{X} \to \mathbf{Y}$ and $h^*\colon \mathbf{Y} \to 2^{\mathbf{X}}$ will vary with the source image and would thus be data-dependent. As a result the $\mathbf{y}$-th blocks in the corresponding compressed images $\mathbf{a}_c, \mathbf{b}_c \in \mathbf{G}^{\mathbf{Y}}$ may or may not have partially or totally overlapping domains. We call this *non-isomorphic encoding*, which is usually employed in conjunction with variable blocksize. Depending on the shape or size of the encoding blocks and whether or not they overlap, the following salient cases pertain to the design of operations over $\mathbf{a}_c$ and $\mathbf{b}_c$:

Case 1. Let the $\mathbf{y}$-th blocks in $\mathbf{a}$ and $\mathbf{b}$ have domains that are a subset of $\mathbf{X}$, which are denoted by $h^*(\mathbf{y})$, where $\mathbf{y} \in \mathbf{Y}$. Let such domains be disjoint or partially overlapping. Given $\mathbf{a}_c(\mathbf{y})$, one must search $\mathbf{b}_c$ to find one or more blocks $\mathbf{b}_c(\mathbf{z})$, $\mathbf{z} \in \mathbf{Y}$, hat contain information about $\mathbf{b}|_{h^*(\mathbf{y})}$.

Case 2. If h and h^* are invariant to $\mathbf{a}$ and $\mathbf{b}$, then Observation 3.4.2.24 applies.

3.4.2.26. Remark. If Case 1 of Observation 3.4.2.25 holds, then search over $\mathbf{b}_c$ requires between one and $|\mathbf{Y}| - 1$ search operations. If search occurs for each block indexed by $h(\mathbf{X})$, then $\mathbf{O}(|\mathbf{Y}|^2)$ search overhead is associated with a binary operation over imagery compressed by T_{LN}. Recall that if $W_{\bigcirc'}(\mathbf{Y}) < W_{\bigcirc}(\mathbf{X})$, then $\eta > 1$ and a computational efficiency accrues from compressive processing. Implementationally, this means that if an average of M blocks must be searched for each of $|\mathbf{Y}|$ invocations of a binary operation, then we have the condition that $\eta > 1$ if and only if $\mathrm{M} \cdot |\mathbf{Y}| > |\mathbf{X}|$. From the definition of $\mathrm{CR_d}$, that means $\mathrm{M} < \mathrm{CR_d}$ is required. When a proportionality constant c that depicts the difference in work required by operations over $\mathbf{Y}$ (versus operations over $\mathbf{X}$) is included, we have $\mathrm{Mc} \cdot |\mathbf{Y}| > |\mathbf{X}|$, which implies that $\mathrm{M} < \mathrm{CR_d}/\mathrm{c}$ if $\eta > 1$ is to be achieved.

3.4.2.27. Remark. The preceding discussion has practical applications, since variable-blocksize transforms are found in the literature (e.g., runlength encoding, variable-blocksize VQ, fractal-based encoding). Since encoding block configurations can vary widely and can be nonrectangular, the analysis of computational efficiency can be involved. Thus, we reserve discussion of specific algorithms for block search and variable blocksize encoding until Parts II and III of this dissertation.

We next consider the case of global reduce operations over compressed imagery, which can be simpler than the binary pointwise operations.

3.4.3. Global Reduce Operations.

We begin with observations about several straightforward cases.

3.4.3.1. Assumption. Let a CCS $X_\Gamma = (\{\mathbf{F}^\mathbf{X}, \mathbf{F}; \Gamma\}, \{T, I; \Theta\}, \{\mathbf{G}^\mathbf{Y}, \mathbf{F}; \Gamma'\})$ be based upon a global reduce operation $\Gamma: \mathbf{F}^\mathbf{X} \to \mathbf{F}$ that has a dual $\Gamma': \mathbf{F}^\mathbf{X} \to \mathbf{G}$ over the range space of $T: \mathbf{F}^\mathbf{X} \to \mathbf{G}^\mathbf{Y}$.

3.4.3.2. Observation. Implicit in Assumption 3.4.3.1 is the fact that all values in $\mathbf{F}$ may not be represented by corresponding values in $\mathbf{G}$. For example, if $\mathbf{a} \in \mathbf{F}^\mathbf{X}$, then $\Sigma\mathbf{a}$ might be in $\mathbf{F}$ but not in $\mathbf{G}$ for one or more of the following implementational reasons:

(i) Each value in $\mathbf{G}$ may be a scalar that represents a vector or encoding block constructed of typical source pixel values in $\mathbf{F}$. For example, since $\Sigma\mathbf{a}$ is a scalar, $\Sigma\mathbf{a}$ might not be represented by a value in $\mathbf{G}$.

(ii) Values in $\mathbf{G}$ may represent a subset of values in $\mathbf{F}$, but $\mathbf{G}$ may not represent all possible source vector configurations (e.g., as in vector quantization). For example, $\Sigma\mathbf{a}$ may not be represented in $\mathbf{G}$.

Additionally, we could have the trivial cases where $\Gamma\mathbf{a} \notin \mathbf{F}$ (e.g., if $\mathbf{F} = \{0,1\}$ and $\Gamma = \Sigma$) or an implementation of $\Gamma: \mathbf{F}^\mathbf{X} \to \mathbf{F}$ is not known.

3.4.3.3. Technique. Let Assumption 3.4.3.1 hold and assume (per Observation 3.4.3.2) that a value in *range*(Γ) may not be represented in G. If $\Gamma \in \{\Sigma, \Pi, \vee, \wedge\}$, then we state the following simple cases:

Case 1. If $T: \mathsf{F}^k \rightarrow \mathsf{G}$ is an exact block encoding, then each block in *domain*(T) has one and only one label in G. As a result, we can build a dual of Γ by applying Γ to each encoding block to produce a value in F. We then construct a many-to-one map $g: \mathsf{F} \rightarrow \mathsf{G}$ that encodes the result of Γ as a value or group of values in G. From the output of g, we can build a lookup table that implements Γ' over values in G.

Case 2. Case (i) may apply where T is a lossy encoding, such as VQ. In such cases, g and a dual Γ' can be constructed. However, $\Gamma'(T(\mathbf{a}))$ may not equal $\Gamma(\mathbf{a})$, i.e., may be an approximate dual of Γ, since the values in *range*($T(\mathbf{a})$) over which Γ operates are approximations of blocks in *domain*(T).

Case 3. Let T be a spatial warping transform that isomorphically minifies or magnifies the source image by a factor K. If $\Gamma \in \{\Sigma, \vee, \wedge\}$, then $\Gamma'(T(\mathbf{a})) = \Gamma(\mathbf{a})/\mathrm{K}$. Likewise, if $\Gamma = \Pi$ and *domain*($\mathbf{a}$) is m-dimensional, then $\Gamma'(T(\mathbf{a})) = \Gamma(\mathbf{a})/\mathrm{K}^m$. In the former case, we multiply the output of Γ' by K to obtain $\Gamma(\mathbf{a})$, and the latter case is stated symmetrically.

Case 4. In a few rare cases, T can preserve certain $\Gamma(\mathbf{a})$ in the compressed image $T(\mathbf{a})$. For example, T could have a quantization step that finds the maximum $\vee\mathbf{a}$ of in image in *domain*(T), then clamps a range of values less than $\vee\mathbf{a}$ to the image maximum. In such cases, T could preserve $\vee\mathbf{a}$ in *range*($T(\mathbf{a})$). Then, the dual of $\Gamma = \vee$ would entail locating $\vee\mathbf{a}$ as it is represented in *range*($T(\mathbf{a})$).

The following examples are illustrative.

3.4.3.4. Example. Let Assumption 3.4.3.1 hold, and let T form a representation of a source encoding block in image $\mathbf{a}$ such that the block mean is preserved in *range*($T(\mathbf{a})$). If $\Gamma \in \{\Sigma, \Pi, \vee, \wedge\}$, then the approximate dual $\Gamma' = \Gamma$ can be applied to $T(\mathbf{a})$ to obtain an approximation to $\Gamma(\mathbf{a})$. We utilize this method in our development of operations over the range spaces of BTC, VQ, and VPIC transformations.

3.4.3.5. Example. Let Assumption 3.4.3.1 hold, and let T form a representation of a source encoding block $\mathbf{b} \in \mathbf{F}^k$ in image $\mathbf{a}$ such that one or more such blocks are represented by an exemplar in a codebook $\mathbf{c} \in (\mathbf{F}^k)^{\mathbf{U}}$, where $\mathbf{U}$ is an indexing set. Given a compressed image $\mathbf{a}_c = T(\mathbf{a}) \in \mathbf{U}^{\mathbf{Y}}$ whose histogram is given by $\mathbf{h} \in \mathbf{N}^{\mathbf{U}}$, the following statements hold:

(i) If $\Gamma = \Sigma$, then

$$\Gamma'(\mathbf{a}_c) = \sum_{g \in G} \mathbf{h}(g) \cdot \Sigma \mathbf{c}(g) \approx \Sigma \mathbf{a}. \tag{108}$$

(ii) If $\Gamma = \vee$, then

$$\Gamma'(\mathbf{a}_c) = \vee \mathbf{c} \approx \vee \mathbf{a}. \tag{109}$$

A symmetric case holds for image minimum.

(iii) If $\Gamma = \Pi$, then

$$\Gamma'(\mathbf{a}_c) = \prod_{g \in G} [\Pi \mathbf{c}(g)]^{\mathbf{h}(g)} \approx \Pi \mathbf{a}. \tag{110}$$

The preceding technique is useful for deriving operations that process over VQ-compressed imagery.

3.4.3.6. Observation. In the preceding example, if the codebook has M exemplars of size k pixels, then Mk operations are required to compute the approximate dual of $\Gamma(\mathbf{a})$. If M is small (customarily less than 256 exemplars, in the case of high-compression VQ), then $M \ll |\mathbf{Y}|$. For example, if 4×4-pixel encoding blocks are employed, then $k = 16$ and $Mk < 4096$ operations. If $|domain(\mathbf{a})| = 1M$, then $|\mathbf{Y}| = 64K$, which far exceeds M. In such cases, the work required would be far less than $|\mathbf{Y}|$ operations, and the computational efficiency would greatly exceed CR_d.

3.4.3.7. Remark. The preceding techniques cover the majority of instances where we derive analogues or duals of global reduce operations over the range spaces of transforms in Classes 1–4. In this study, the following exceptions hold:

(i) Nonlinear inner product transforms, which generally do not preserve a representation of $\Sigma\mathbf{a}$ in $T(\mathbf{a})$;

(ii) Encryptions that unpredictably represent the number of unitary bits present in $\mathbf{a}$. From the cryptographic perspective, such encodings are useful since the correlation between $\Sigma\mathbf{a}$ and ciphertext $T(\mathbf{a})$ is thus reduced, thereby reducing disclosure of $\mathbf{a}$'s salient properties to an adversary; and

(iii) Transformations (mainly encryptions) that represent numbers in $\mathbf{a}$ with information loss as values in $T(\mathbf{a})$ whose correspondence with source values is difficult to trace. Such ciphers belong to a class of encryptions called *steganography* (i.e., hidden writing) and are not considered in this study due to their complexity.

We next consider techniques for deriving analogues and duals of image-template operations.

3.4.4. Image-Template Operations.

There are four primary methods with which image-template operations can be simulated over the range space of a compressive or encryptive transform. For example, one can use the image-domain template in compressed or uncompressed form. Additionally, the analogous or dual operation can be derived analytically or, in certain cases, can be encoded in a lookup table.

3.4.4.1. Assumption. Let the CCS

$$X_{\textcircled{T}} = \left(\left\{\mathbf{F}^{\mathbf{X}}, (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}};\ \textcircled{T}\right\},\ \{T,\ U;\ \Theta\},\ \left\{\mathbf{G}^{\mathbf{Y}}, (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}};\ \textcircled{T}'\right\}\right)$$
$$V_{\textcircled{T}} = \left(\left\{\mathbf{F}^{\mathbf{X}}, (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}};\ \textcircled{T}\right\},\ \{T,\ I;\ \Theta\},\ \left\{\mathbf{G}^{\mathbf{Y}}, (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}};\ \textcircled{T}'\right\}\right), \tag{111}$$

where $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, the template transform $U: (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}$, I denotes the identity transform, and $\textcircled{T}' = \Theta(\textcircled{T})$, with $\textcircled{T}: \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$.

3.4.4.2. Assumption. Let Assumption 3.4.4.1 hold, and let source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, compressed (encrypted) image $\mathbf{a}_c = T(\mathbf{a})$, and computed image $\hat{\mathbf{a}} = \mathbf{a}\ \textcircled{T}\ \mathbf{t}$.

3.4.4.3. Technique. Let Assumptions 3.4.4.1 and 3.4.4.2 hold. As mentioned in the introductory comments to this section, the following four cases are pertinent to the derivation of analogues or duals of $\textcircled{T}$ over *range*(T):

Case 1. If U exists, then the operation $\textcircled{T}$ can be simulated over *range*(T) by $\textcircled{T}'$ that accepts $\mathbf{a}_c$ and $\mathbf{s} = U(\mathbf{t})$, and produces $T(\mathbf{a}\ \textcircled{T}\ \mathbf{t})$ or an approximation thereof. This is accomplished with the CCS $X_{\textcircled{T}}$. Note that $\textcircled{T}'$ can be implemented in terms of a lookup table that accepts values in $\mathbf{a}_c$ within the support of $\mathbf{s}$, and produces the value in $T(\mathbf{a}\ \textcircled{T}\ \mathbf{t})$ that occurs at the target point of $\mathbf{s}$, or produces an approximation to that value. Problems and pitfalls associated with this approach are discussed below.

Case 2. The preceding case applies, but $\textcircled{T}'$ is a dual of $\textcircled{T}$, i.e., computes $\mathbf{a}\ \textcircled{T}\ \mathbf{t}$ given $\mathbf{a}_c$ and $\mathbf{s}$.

Case 3. If U does not exist, then the operation $\textcircled{T}$ can be simulated over *range*(T) by $\textcircled{T}'$, which accepts $\mathbf{a}_c$ and $\mathbf{t}$, and produces $T(\mathbf{a}\ \textcircled{T}\ \mathbf{t})$ or an approximation thereof. This is accomplished with the CCS $V_{\textcircled{T}}$, and is particularly useful for block-structured operations over block-encoded imagery (e.g., block averaging, smoothing, or edge detection). In such cases, source blocks are preserved either in $\mathbf{a}_c$ or, more usually,

in a codebook. For example, in the case of a codebook $\mathbf{c}$, template $\mathbf{t}$ can be applied to exemplars in $\mathbf{c}$ that are images on a subset of $\mathbf{X}$.

Case 4. The preceding case applies, but $\textcircled{\gamma}'$ is a dual of $\textcircled{\gamma}$, i.e., computes $\mathbf{a}\ \textcircled{\gamma}\ \mathbf{t}$ given $\mathbf{a}_c$ and $\mathbf{t}$.

Although there are additional methods of simulating image-template operations over *range*(T), the preceding four cases cover the majority of derivations encountered in this study. For purposes of illustration, we discuss one instance each of Case 1 and Case 3 in the following observations.

3.4.4.4. Observation. Consider the problem of formulating template transformation U in Case 1 of the preceding section. The obvious approach is to assume that $\mathbf{t}$ is spatially invariant, then transform $\mathbf{t}_\mathbf{x}$, the weight matrix of $\mathbf{t}$, where $\mathbf{x}$ is a source domain point in $\mathbf{X}$. We thus produce the weight matrix of the compressive template $\mathbf{s}$ as $\mathbf{s}_{h(\mathbf{x})} = T(\mathbf{t}_\mathbf{x})$.

Several practical problems are associated with this approach. We first assume, for purposes of convenience, that T is a blockwise transform which employs constant block shape and size. Additionally, we assume (for purposes of notational convenience) that the analogous operation $\textcircled{\gamma}'$ is comprised of two associative commutative operations $\bigcirc', \gamma' : \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$. If $S(\mathbf{t}_\mathbf{x})$ subtends an integral number N of encoding blocks, then U employs T to transform $\mathbf{t}_\mathbf{x}$ into an array or list of N values in $\mathbf{G}$, each of which represents an encoding block. Typically, no further manipulation is required.

Unfortunately, there are numerous cases where $S(\mathbf{t}_\mathbf{x})$ does not completely subtend N encoding blocks. In such cases, padding of $\mathbf{t}_\mathbf{x}$ is required. For example, if $\mathbf{W} = h(S(\mathbf{t}_\mathbf{x}))$, and $\mathbf{b}$ is a constant image on $\mathbf{W}$ whose pixel values equal the zero of $\bigcirc'$, then the corresponding weight matrix of $\mathbf{s}$ would be given by

$$\mathbf{s}_{h(\mathbf{x})} = T\left(\left(\mathbf{t}_\mathbf{x}|^{\mathbf{b}}\right)|_{h^*(\mathbf{W})}\right), \mathbf{x} \in \mathbf{X}, \tag{112}$$

where h^* was defined previously.

There are several problems with this approach. First, by zero-padding $\mathbf{t_x}$, one may cause T to distort the representation of encoding blocks in $\mathbf{t_x}$. For example, let T characterize an encoding block of size k pixels by its mean, and assume that a given block partition of $\mathbf{t_x}$, having size n pixels, subtends a portion of the $\mathbf{y}$-th encoding block, where $\mathbf{y}$ is a point in the compressed domain $\mathbf{Y}$. We call such a partition of $\mathbf{t_x}$ an *incomplete block partition*. In such cases, zero-padding will cause the $\mathbf{y}$-th block mean to be reduced by a factor n/k of its value computed over n pixels. In practice, the n-pixel mean portrays the given (incomplete) block partition of $\mathbf{t_x}$ more accurately.

3.4.4.5. Remark. When we consider the problem of template matching over compressed images, we shall see that the preceding issue is not fanciful, but has direct impact upon the accuracy with which partitions that comprise targets of interest can be recognized in compressed imagery. For example, BTC, VPIC, and block-normalized VQ characterize encoding blocks in terms of the block mean. Thus, we can either pad incomplete block partitions of $\mathbf{t_x}$ that characterize targets of interest with typical (e.g., mean) background values, or we can truncate incomplete partitions from $\mathbf{t_x}$ prior to its transformation into the weight matrix of $\mathbf{s}$. Background padding, although attractive conceptually, has the disadvantage of rendering $\mathbf{t}$ specific to one type of target-background configuration. Likewise, truncation may incur inaccuracies in representing target shape or statistics. We discuss these methods briefly in the final portion of this dissertation and plan to explore alternative techniques in future research.

3.4.4.6. Complexity. The method of Case 1 requires $\mathbf{O}(|\mathbf{Y}|^2)$ work, since both the source image and template are compressed. Given that $\mathbf{O}(|\mathbf{X}|^2)$ work is required for noncompressive image-template operations (over $\mathbf{X}$) this represents a net efficiency of $\mathbf{O}(\mathrm{CR_d}^2)$, which exceeds the study goal of $\mathbf{O}(\mathrm{CR_d})$ efficiency. Unfortunately, if $\mathbf{t}$ is space-variant, then this efficiency is severely degraded by the overhead incurred by transformation

of the weight matrix $\mathbf{t}_{\mathbf{x}}$ at each point $\mathbf{x} \in \mathbf{X}$. If we assume, for purposes of convenience, that $\mathbf{t}$ is invariant within each encoding block, then $|\mathbf{Y}|$ such transformations are required.

3.4.4.7. Observation. An alternative to Case 1 is given by Case 3, where we could combine the source template $\mathbf{t}$ with a codebook exemplar or compressed block representation. Since the codebook processing approach is more typical in current practice, we let Assumptions 3.4.4.1 and 3.4.4.2 hold, and denote the codebook as $\mathbf{c} \in (\mathbf{F}^{\mathbf{k}})^{\mathbf{U}}$, where each exemplar is a k-pixel, $\mathbf{F}$-valued block that contains the values of an image on a subset of $\mathbf{X}$. Given an exemplar index $\mathbf{u}$ in the codebook domain $\mathbf{U}$, we produce a new exemplar $\mathbf{d}(\mathbf{u})$ by applying $\mathbf{t}$ to $\mathbf{c}(\mathbf{u})$ as

$$\mathbf{d}(\mathbf{u}) = \mathbf{c}(\mathbf{u}) \textcircled{?} \mathbf{t} . \tag{113}$$

By thus processing $|\mathbf{U}|$ exemplars, we produce a new codebook $\mathbf{d}$ from $\mathbf{c}$. If the compressed image $\mathbf{a}_{\mathbf{c}} \in \mathbf{U}^{\mathbf{Y}}$, then processing of $\mathbf{a}_{\mathbf{c}}$ is not required, since Equation 113 redefines the codebook only.

The problem of boundary conditions that occur in Equation 113 can be remedied as follows:

Step 1. Expand in the i-th dimension of $\mathbf{X}$ the corresponding boundaries of each codebook exemplar in $\mathbf{c}$ by m_i pixels, where m denotes the halfwidth of $\mathbf{t}$.

Step 2. Apply Equation 113 to each extended exemplar in $\mathbf{c}$ to yield $\mathbf{d}$.

Step 3. For each exemplar in $\mathbf{d}$, remove the pixels added in Step 1, thereby yielding exemplars of size identical to those in $\mathbf{c}$.

Although there is residual error at the exemplar boundaries, such error is much less than that which would be present if the boundaries were not expanded. That is, when $\mathbf{t}$'s target point coincides with a boundary pixel of a given exemplar, the support of $\mathbf{t}$ overlies pixels that would be artificially valued with the zero of $\bigcirc : \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$, a constituent operation of $\textcircled{?}$.

3.4.4.8. Complexity. Given a codebook of M k-pixel exemplars and an N-pixel source template, Equation 113 requires $\mathbf{O}(MNk)$ operations, as opposed to $\mathbf{O}(N \cdot |\mathbf{X}|)$ operations in the uncompressive case. Since $M \ll |\mathbf{Y}|$ and $Mk < |\mathbf{Y}|$ are typical, supra-unitary computational efficiencies can be achieved.

3.4.4.9. Example. Let a codebook have M = 256 exemplars of 4×4 = 16 pixels each. Let an N-pixel source image be linearly convolved (i.e., $\textcircled{\gamma} = \oplus$) with a source template having 25 pixels, which is zero-padded to 8×8 pixels prior to transformation of its weight matrix (per Equation 3.4.4.4). Thus, the size of the uncompressed template is given by N = 25 pixels, while the compressed template has L = 2×2 = 4 pixels. Assuming that $\mathbf{X}$ is a 1024×1024-pixel array and $\mathbf{N} \cdot |\mathbf{X}|$ multiplications are required for noncompressive convolution, and assuming that the noncompressive multiplication operation can be used in the compressive convolution, the computational efficiency is given by

$$\eta = \frac{N \cdot |\mathbf{X}|}{M \cdot k \cdot L} = \frac{25 \cdot 1024^2}{256 \cdot 16 \cdot 4} = 1600, \tag{114}$$

which is nontrivial. Even if the analogue of convolution requires operations that are 16 times more costly than the noncompressive multiplications, a speedup of two orders of magnitude would remain, which greatly exceeds the domain compression ratio of 16:1.

3.4.4.10. Observation. An alternative to combining a compressed image $\mathbf{a}_c$ and a compressed template $\mathbf{s}$ on $domain(\mathbf{a}_c)$ using $\textcircled{\gamma}$ can be obtained via a lookup table. Due to current constraints on memory size, this method is useful only for small templates convolved over images whose greylevels are quantized into an interval A such that $siz(A)$ is small.

In practice, we prefer to use invariant templates to avoid LUT reconfiguration. In such cases, we merely configure the LUT domain to accept values in $\mathcal{S}(\mathbf{s}_\mathbf{y})$, where $\mathbf{y}$ is a point in $domain(\mathbf{a}_c)$. The LUT then outputs the template value at the target point $\mathbf{y}$.

3.4.4.11. Complexity. Let $\mathbf{a}_c$ be produced by BTC, such that the $\mathbf{y}$-th compressed image value contains the mean μ, standard deviation σ, and a zero crossing bitmap $\mathbf{d}$ of the $\mathbf{y}$-th encoding block, where $\mathbf{y}$ is a point in $domain(\mathbf{a}_c)$. Let the compressed template $\mathbf{s}$ be the unitary von Neumann template, for purposes of simplicity. For purposes of brevity, we assume that $\mathbf{s}$ operates upon the block mean only, and let μ be quantized into four bits, which is permissible in BTC of noisy imagery. Since $|S(\mathbf{s}_\mathbf{y})| = 5$, the LUT has a domain size of 20 bits, which implies 1M elements storage. Since $\mathbf{s}$ operates upon the mean only, $\mathbf{s}$ returns a mean value (4 bits), which implies a total LUT size of 500KB, which is feasible implementationally.

In Chapter 9, we show that local averaging can be simulated over BTC- and VPIC-compressed imagery by such methods, with a consequent cost of $\mathbf{O}(|\mathbf{Y}|)$ operations, where the proportionality constant slightly $|S(\mathbf{s}_\mathbf{y})|$ due to I/O overhead and data structural manipulations.

Given the preceding theoretical background, we next consider the more specialized cases of operations over the range space of specific transform classes in our taxonomy.

3.4.5. Analogues over the Range of Class 1 Transforms.

We describe two techniques for Class 1 transforms, which are (a) the definition of $\bigcirc'$ in terms of $\bigcirc$, and (b) the use of isomorphisms between $\bigcirc$ and $\bigcirc'$.

3.4.5.1. Techniques.

Method 1. *Expressing* $\bigcirc'$ *in terms of* $\bigcirc$. Let Assumption 3.4.1.1 hold, and let T_1: $\mathbf{F}^\mathbf{X} \rightarrow \mathbf{F}^\mathbf{X}$ be a homomorphism from S to S. From Assumption 3.4.1.1, it follows that $\bigcirc, \bigcirc'$: $S \times S \rightarrow S$. Since $\bigcirc$ and $\bigcirc'$ have the same range and domain, it is reasonable to assume that $\bigcirc'$ could be expressed in terms of $\bigcirc$ or (in restricted instances) that $\bigcirc' = \bigcirc$. For example, in the case of substitutional ciphers that are bijections, $\bigcirc'$ can be expressed in terms of $\bigcirc$.

Method 2. Deriving $\bigcirc'$ analytically from $\bigcirc$ and T. Assume that $T_1\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ is a homomorphism, $\bigcirc', \bigcirc\colon S \times S \to S$, and source images $\mathbf{a},\mathbf{b} \in \mathbf{F}^{\mathbf{X}}$. From the definition of a homomorphism, $T_1(\mathbf{a} \bigcirc \mathbf{b}) = T_1(\mathbf{a}) \bigcirc' T_1(\mathbf{b})$. Since T, $\mathbf{a}$, $\mathbf{b}$, and $\bigcirc$ are known, instances of the preceding equation may be soluble for $\bigcirc'$. For example, the solution is generally easier for customary local (e.g., pointwise) transformations than for global (e.g., inner-product) transforms.

The following examples are illustrative.

3.4.5.2. Example: Method 1. Let $T_1\colon \mathbf{R}^{\mathbf{X}} \to \mathbf{R}^{\mathbf{X}}$, such that $T_1(\mathbf{a}) = 2 * \mathbf{a}$, a pointwise transformation. Let $\bigcirc\colon \mathbf{R}^{\mathbf{X}} \times \mathbf{R}^{\mathbf{X}} \to \mathbf{R}^{\mathbf{X}}$ be defined as $\bigcirc(\mathbf{a},\mathbf{b}) = \mathbf{a} + \mathbf{b}$. Then, $\bigcirc' = \bigcirc$, since multiplication distributes over addition.

3.4.5.3. Example: Method 2. Let $\mathbf{X} = \mathbf{Z}_n$ and let the homomorphism $T_1\colon (\mathbf{Z}_2)^{\mathbf{X}} \to (\mathbf{Z}_2)^{\mathbf{X}}$, with $\bigcirc, \bigcirc'\colon (\mathbf{Z}_2)^{\mathbf{X}} \times (\mathbf{Z}_2)^{\mathbf{X}} \to (\mathbf{Z}_2)^{\mathbf{X}}$. Define the transform $T_1(\mathbf{a}) = \neg(\mathbf{a}) = (\mathbf{a} + 1) \bmod 2$ and let the image operation $\bigcirc(\mathbf{a},\mathbf{b}) = \mathbf{a} \text{ XOR } \mathbf{b} = (\mathbf{a} + \mathbf{b}) \bmod 2$. Given DeMorgan's laws and the definition of homomorphism, we can rewrite the expression $T_1(\mathbf{a} \bigcirc \mathbf{b}) = T_1(\mathbf{a}) \bigcirc' T_1(\mathbf{b})$ as

$$\neg(\mathbf{a}) \bigcirc' \neg(\mathbf{b}) = \neg(\mathbf{a} \text{ XOR } \mathbf{b}) = \neg(\mathbf{a}) \,\overline{\text{XOR}}\, \neg(\mathbf{b}), \tag{115}$$

where $\mathbf{a} \,\overline{\text{XOR}}\, \mathbf{b} = (\mathbf{a} - \mathbf{b}) \bmod 2$. Thus, $\bigcirc' = \overline{\text{XOR}}$. Given Definitions 3.2.2.7 and 3.2.3.12, together with the Boolean operators $\mathcal{U}_{\mathbf{Z}_2}$, we therefore have the following solution to an instance of the TPP

$$\mathrm{M}_{\text{TPP}}\colon \left(\left((\mathbf{Z}_2)^{\mathbf{X}}, \text{XOR}\right), \neg, \left((\mathbf{Z}_2)^{\mathbf{X}}, \mathcal{U}_{\mathbf{Z}_2}\right)\right) \vdash \left((\mathbf{Z}_2)^{\mathbf{X}}, \overline{\text{XOR}}\right), \tag{116}$$

which yields the computational system

$$X_{\bigcirc} = \left(\{\mathbf{Z}_2^{\mathbf{X}}, \text{XOR}\}, \{\neg, \neg\}, \{\mathbf{Z}_2^{\mathbf{X}}, \overline{\text{XOR}}\}\right). \tag{117}$$

In practice, many Class 1 transforms are amenable to such simple derivational methods. The derivation of an analogous or dual operation is nearly trivial when T_1 is a linear transform and $\bigcirc$ is a linear operator. However, when $\bigcirc$ is nonlinear, an approximation is generally required (per Technique 2 of Section 3.4.5.1). Similar methods are employed for global-reduce and image-template operations, as shown in the derivation of TPP solutions for Class 1 transforms (reference Chapter 6). We next consider the determination of analogous operations over the outputs of Class 2 transforms.

3.4.6. Analogues over the Range of Class 2 Transforms.

Class 2 transforms modify the range space of an image but not the image domain, and can be thought of implementationally as a subset of the greyscale operations.

3.4.6.1. Observation. Note that certain operations over *domain*(T_2) can be transformed into operations over *range*(T_2) by the application of a greyscale transform. For example, bijective transpositions over a uniquely valued image are isomorphic to value substitutions. Alternatively, T_2 can be a many-to-one mapping such as a local neighborhood transform (e.g., local averaging or median filtering followed by bit-slicing). The Class 2 transform $T_2 : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{X}}$ can be thought of as a scalar transform ${}^{\mathrm{M}}T_2 : \mathbf{F}^{\mathbf{n}} \rightarrow \mathbf{G}$. Note that ${}^{\mathrm{M}}T_2$ admits n-ary kernel functions that compress data by grouping several neighborhood values in terms of a value in $\mathbf{G}$, similar to vector quantization.

Similarly, at fine granularity, we think of a Class 2 transform as a grey-level mapping ${}^{\mathrm{F}}T_2 \equiv g : (f_1, f_2, \ldots, f_n) \mapsto g$, where $f_1, f_2, \ldots, f_n \in \mathbf{F}, g \in \mathbf{G}$. When T_2 is not invertible, an closed-form solution may not be obtained by solving the definition of an isomorphism (using T_2 and $\bigcirc$) for the analogous operation $\bigcirc$. Note that when n is large, search over $|\mathbf{F}|^{\mathbf{n}}$ incurs exponentially-growing cost that becomes prohibitively large for practical values of n.

As a result, neighborhood-based formulations of T_2 usually have no ϵ-near approximation to the inverse transform that can be obtained by efficient search over $|\mathbf{F}|^{\mathbf{n}}$. This fact

has adverse implications for cryptanalysis. Similarly if a VQ-based approach to solving for $\bigcirc'$ is employed (by attempting to map clusters of values in $|\mathbf{F}|^n$ to corresponding exemplars), then if n is large, $|\mathbf{F}|^n$ may be so sparsely populated that quantization yields a prohibitive distance (or error) between an exemplar and the data it represents. As a result of the foregoing problems, we concentrate (for purposes of brevity) on tractable pointwise instances of T_2, and defer discussion of approximations of $\bigcirc'$ to subsequent development.

3.4.6.2. Technique. If T_2 is a homomorphism, then the statement $T_2(\mathbf{a} \bigcirc \mathbf{b}) = T_2(\mathbf{a}) \bigcirc' T_2(\mathbf{b})$ can be written as $g(\mathbf{a} \bigcirc \mathbf{b}) = g(\mathbf{a}) \bigcirc' g(\mathbf{b})$. If g and $\bigcirc$ can be expressed analytically, then the preceding equation can be solved for $\bigcirc'$, within constraints of tractability.

When T_2 is a bijection, it is convenient to implement an analogue $\bigcirc'$ of an image operation $\bigcirc$ in terms of the inverse mapping $g^{-1}: \mathbf{G} \rightarrow \mathbf{F}$. For example, given a unary image operation $\bigcirc: \mathbf{F} \rightarrow \mathbf{F}$, the commutativity diagram of Figure 1 implies that $\bigcirc'(g) = \bigcirc(g^{-1}(g))$, $g \in \mathbf{G}$. As discussed in Chapter 4, such formulations exhibit the disadvantage of decreased data security, since a bijection is an isomorphism and g^{-1} can disclose information about $\mathbf{F}$.

The following simple examples illustrates the concept of deriving an operation over *range*(T_2). More detailed examples are given in Chapter 7.

3.4.6.3. Example. Let the homomorphism $T_2: (\mathbf{R}^+)^{\mathbf{X}} \rightarrow \mathbf{R}^{\mathbf{X}}$, such that $T_2(\mathbf{a}) = \sin(\mathbf{a})$. Let $\bigcirc: (\mathbf{R}^+)^{\mathbf{X}} \times (\mathbf{R}^+)^{\mathbf{X}} \rightarrow (\mathbf{R}^+)^{\mathbf{X}}$ be defined as $\bigcirc(\mathbf{a},\mathbf{b}) = \mathbf{a} + \mathbf{b}$. An analogous operation $\bigcirc'(T(\mathbf{a}), T(\mathbf{b})) = T(\bigcirc(\mathbf{a},\mathbf{b})) = \sin(\mathbf{a} + \mathbf{b})$ is given by

$$\begin{aligned} \bigcirc'(\sin \mathbf{a}, \sin \mathbf{b}) &= \sin \mathbf{a} \cdot \cos \mathbf{b} + \sin \mathbf{b} \cdot \cos \mathbf{a} \\ &= \sin \mathbf{a} \cdot \sin\left(\mathbf{b} + \frac{\pi}{2}\right) + \sin \mathbf{b} \cdot \sin\left(\mathbf{a} + \frac{\pi}{2}\right), \end{aligned} \tag{118}$$

which follows from basic trigonometry.

We next provide an example of deriving an operation $\bigcirc'$ over the range space of a many-to-one Class 2 transform, which operation is not an analogue of the corresponding

image operation $\bigcirc$. In particular, $\bigcirc$ yields an image that has certain pixels that are a predictable subset of pixels in the output of $\bigcirc'$.

3.4.6.4. Example. Consider the *reduction cipher* $T_r: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{X}}$, where $|\mathbf{F}| > |\mathbf{G}|$. For example, a message in English plaintext (where $|\mathbf{F}| = 26$) could map with two-fold size reduction to a message in a language where $|\mathbf{G}| = 13$ (e.g., $\mathbf{G} = \{a,e,i,o,u,h,k,l,m,n,p,w,y\}$). By choosing the mapping appropriately, a particular plaintext word or phrase could be translated into legible ciphertext, while the remainder of the ciphertext could be gibberish. This method is similar to that of *steganography*, which encodes text messages in pictures formed from ASCII characters. Alternatively, one could pair frequently occurring characters with infrequent characters, in order to render the frequency distribution of encoding symbols more uniform.

In a simple case, if T_r does not perturb the message ordering, then an operation that preserves message ordering and does not perturb symbol values (e.g., concatenation), would be preserved over *range*(T_r). In a more complicated case, if an N-pixel plaintext template was encoded using T_r, where N is an integral multiple of the number of pixels in each of T_r's input values, then the operation of template matching would not be preserved over *range*(T_r), since T_r is a many-to-one mapping.

Let us examine the latter assertion by setting $\mathbf{F} = \{A,B,C,D\}$ and $\mathbf{G} = \{0,1\}$, for purposes of simplicity. Suppose that T_r maps A or B to 0 and C or D to 1. Given the message *ABACAD*, T_r would produce the ciphertext *000101*. Let a template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, and let a template matching operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ produce D whenever $(\mathbf{a}|_{\mathcal{S}(\mathbf{t}_\mathbf{y})})(\mathbf{x}) = \mathbf{t}_\mathbf{y}(\mathbf{x})$, $\mathbf{x},\mathbf{y} \in \mathbf{X}$, and an A otherwise. Assume that $\mathbf{t}$ can be encoded as a template $\mathbf{s} \in (\mathbf{G}^{\mathbf{X}})^{\mathbf{X}}$ by the application of T_r to the weight matrix $\mathbf{t}_\mathbf{y}$, such that $\mathbf{s}$ is an exact encoding of $\mathbf{t}$. Let $\bigcirc': \mathbf{G}^{\mathbf{X}} \times (\mathbf{G}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{X}}$ match $\mathbf{s}$ to $\mathbf{a}_c = T_r(\mathbf{a})$, producing $\mathbf{a}_c(\mathbf{y}) = 1$ whenever $(\mathbf{a}_c|_{\mathcal{S}(\mathbf{s}_\mathbf{y})})(\mathbf{x}) = \mathbf{s}_\mathbf{y}(\mathbf{x})$, $\mathbf{x},\mathbf{y} \in \mathbf{X}$, and zero otherwise. Since T_r is a

many-to-one mapping, $T_r(\bigcirc(\mathbf{a},\mathbf{t})) \neq \bigcirc'(\mathbf{a}_c,\mathbf{s})$ and $\bigcirc'$ is not an analogue of $\bigcirc$. However, $T_r(\bigcirc(\mathbf{a},\mathbf{t}))\|_{\{1\}} \subset \bigcirc'(\mathbf{a}_c,\mathbf{s})$.

This can be verified by setting $\mathbf{a} = (A,B,B,A,B,C,A,C,D,A)$ and $\mathbf{t} = (A,\underline{B})$, where the target pixel is underlined, then $\mathbf{a}_c = (0,0,0,0,0,1,0,1,1,0)$ and $\mathbf{s} = (0,\underline{0})$. Thus,

$$\bigcirc(\mathbf{a},\mathbf{t}) = (A, D, A, A, D, A, A, A, A, A),$$

$$T_r(\bigcirc(\mathbf{a},\mathbf{t})) = (0, 1, 0, 0, 1, 0, 0, 0, 0, 0), \tag{119}$$

$$\text{and } \bigcirc'(\mathbf{a}_c,\mathbf{s}) = (0, 1, 1, 1, 1, 0, 0, 0, 0, 0) \neq T_r(\bigcirc(\mathbf{a},\mathbf{t})).$$

As a result, analogues of many image operations often do not exist over the range spaces of many-to-one mappings in Class 2. However, as shown in Chapter 7 and outlined in the following section, it is possible in certain cases to perturb the original image operation, or to utilize the concept of an ϵ-near approximation to the inverse transform to obtain an analogue over *range*(T_2).

We next consider processing over the range spaces of the Class 3 transforms, which was summarized in Section 3.4.2.11.

3.4.7. Analogues over the Range of Class 3 Transforms.

In the Class 3 transforms (denoted by T_3), the domains of the input and output images are unequal. Thus, we can think of the Class 3 transforms as *domain transformations.* If the image domain is a subset of $\mathbf{R}^n$, then such mappings are frequently called *spatial transforms.* As a result of T_3's conceptual and formulaic simplicity, the derivation of analogues over the Class 3 transforms can be accomplished with relative ease.

3.4.7.1. Observation. The Class 3 transforms, of form $T_3: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$, can be thought of as being based upon a spatial mapping $f: \mathbf{Y} \rightarrow \mathbf{X}$. The inverse spatial transform $f^{-1}: \mathbf{X} \rightarrow \mathbf{Y}$, if it exists, can be the basis for a transform-regime analogue of an image operation over *range*(T_3). This concept is similar to our use (for Class 2 transforms) of the inverse $g^{-1}: \mathbf{G} \rightarrow \mathbf{F}$ of the grey-level transform $g: \mathbf{F} \rightarrow \mathbf{G}$ in deriving corresponding analogues over *range*(T_2).

3.4.7.2. Technique. Assume that the Class 3 transforms are represented generally by a spatial transform that is a homomorphism. That is, given $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and the spatial transform $f: \mathbf{Y} \rightarrow \mathbf{X}$, the transformed image $\mathbf{a}_c = T(\mathbf{a}) = \mathbf{a} \circ f$, which composition was defined previously. If f has an inverse $f^{-1}: \mathbf{X} \rightarrow \mathbf{Y}$, let the inverse homomorphism $T^{-1}: \mathbf{F}^{\mathbf{Y}} \rightarrow \mathbf{F}^{\mathbf{X}}$ exists, such that

$$\mathbf{a} = T^{-1}(\mathbf{a}_c) = \mathbf{a}_c \circ f^{-1} = \mathbf{a} \circ f \circ f^{-1}. \tag{120}$$

As shown in Section 3.4.2.11, the analogous operation over *range*(T), denoted by $\bigcirc': \mathbf{F}^{\mathbf{Y}} \rightarrow \mathbf{F}^{\mathbf{Y}}$, can be derived from the preceding equation as

$$\bigcirc'(\mathbf{a}_c) = \bigcirc(\mathbf{a}) \circ f = (\bigcirc \circ f)(\mathbf{a}), \tag{121}$$

since $\mathbf{a}_c = \mathbf{a} \circ f$ and composition is associative.

Thus, if T is a homomorphism and $\bigcirc$ is specified, then $\bigcirc' = \bigcirc \circ f$. A symmetric case holds for binary operations.

3.4.7.3. Remark. Let $\mathbf{a},\mathbf{b} \in \mathbf{F}^{\mathbf{X}}$, assume that T is as given in the preceding section, and let $\mathbf{a}_c = T(\mathbf{a})$ and $\mathbf{b}_c = T(\mathbf{b})$. Without loss of generality, the preceding remark applies to binary pointwise operations as

$$\bigcirc'(\mathbf{a}_c,\mathbf{b}_c) = \bigcirc(\mathbf{a},\mathbf{b}) \circ f = (\bigcirc \circ f)(\mathbf{a},\mathbf{b}), \tag{122}$$

where f was defined previously. Similarly, given a template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, the operation $\mathbf{a} \textcircled{\gamma} \mathbf{t}$ has an analogue $\textcircled{\gamma}'$ over *range*(T) that is comprised of basis operations

$$\bigcirc' = \bigcirc \circ f \qquad \text{and} \qquad \gamma' = \gamma \circ f, \tag{123}$$

which are the analogues over *range*(T) of $\bigcirc, \gamma: \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$, the basis operations of $\textcircled{\gamma}$.

3.4.7.4. Observation. We advance the following comments concerning global reduce and image-template operations over the range spaces of spatial warping transformations, which have obvious proofs.

1. If the spatial transform f is continuous and isomorphic and does not crop or magnify the source image $\mathbf{a}$, then the global reduce functions are analogues of themselves.
2. Let f be discrete and isomorphic, without cropping $\mathbf{a}$, but let f magnify $\mathbf{a}$ by a factor of $r > 0$ in each of m dimensions of *domain*($\mathbf{a}$). Neglecting quantization error, the following statements hold:
   - $\Sigma'\mathbf{a}$, the dual of $\Sigma\mathbf{a}$, is given by $\Sigma'(\mathbf{a}) = r \cdot \Sigma\mathbf{a}$;
   - Image maximum and minimum are self-analogues; and
   - If $\mathbf{a}$ is not unitarily-valued, then $\Pi'\mathbf{a}$, the dual of $\Pi\mathbf{a}$, is given by $\Pi'(\mathbf{a}) = (\Pi\mathbf{a})^{r^m}$;
3. If f is anisomorphic, then analogues or duals of the global reduce functions cannot be predicted globally, but must be computed over small neighborhoods of *range*(f), then combined to yield the functional result.

3.4.7.5. Observation. Image-template operations are computed over *range*(T_3) in a manner that is symmetric to the Class 3 pointwise operations. Assuming an invariant template $\mathbf{t}$, for purposes of convenience, the source image $\mathbf{a}$ and template weight matrix are first transformed by T_3 to yield an image $\mathbf{a}_c$ and a weight matrix of a new template $\mathbf{s}$ that is defined over *domain*($\mathbf{a}_c$). Within the constraints of quantization and interpolation error, the operation $\textcircled{?}$ is a self-analogue.

If f is anisomorphic, then $\mathbf{t}$ (and, hence, $\mathbf{s}$) must be defined as space-variant, such the weight matrix of $\mathbf{s}$ must be computed at each target point $\mathbf{y}$ in *domain*($\mathbf{a}_c$), prior to application of $\textcircled{?}$. This is a key drawback of computing over imagery that is perturbed by spatial warping transforms. We address this issue in some detail in Chapter 8.

In the development of Chapter 9, we present examples of the foregoing technique, and derive various functions of spatially-transformed imagery. We also discuss other types of transformations in Class 3, including those mappings that discard pixels in the transform domain. In such cases, the preceding theory holds for source pixels that remain in the compressed image. However, information loss results, due to the discarding of information from the source image. We next discuss processing over the range spaces of Class 4 transforms.

3.4.8. Analogues over the Range of Class 4 Transforms.

Class 4 contains many interesting and complex compression and encryption transforms. As a result of the formulaic complexity of the transform, analogues of image operations are not as easily derived as for Classes 1–3. However, via the use of substitutions, we can derive general formulations for Class 4 analogues of certain image operations. Additional operations can be formulated via approximation methods.

3.4.8.1. Observation. Recall that, in Chapter 3, we exemplified the multigranular decomposition by specializing Class 4 transforms into two sub-classes. In particular, Class 4–A contained fixed-block encodings that were portrayed as substitutions, and Class 4–B contained variable-block encodings. Note that (a) the medium-granular formulation ${}^{M}T_4\colon (\mathbf{F}^{k})^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ generalizes subclasses 4–A and 4–B and (b) we can represent T_4 in scalar form by the greylevel operation $g\colon (f_1, f_2, \ldots, f_k) \mapsto g$, where $f_1, f_2, \ldots, f_k \in \mathbf{F}$, $g \in \mathbf{G}$, and k may vary with $\mathbf{y} \in \mathbf{Y}$.

This situation is similar to the Class 2 transform, where $g\colon \mathbf{F} \rightarrow \mathbf{G}$. As a result, pointwise image operations and global reduce operations can map easily to the range spaces of Class 4 substitutions, as follows.

3.4.8.2. Technique: Global Reduce Operations. Let the Class 4 transform be decomposed as ${}^{C}T_4 : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}} \triangleq {}^{M}T_4 : (\mathbf{F}^{\mathbf{k}})^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, where ${}^{M}T_4$ is specified in fine-granular form as the function $g : \mathbf{F}^{\mathbf{k}} \rightarrow \mathbf{G}$. For purposes of development, assume that there exists an inverse (or an ϵ-near approximation to the inverse) $g^{-1} : \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{k}}$. Let the source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ be operated upon by a global reduce operation $\Gamma : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}$. Additionally, let the transformed image $\mathbf{a}_c = T_4(\mathbf{a})$, and assume that the encoding blocks denoted by $\mathbf{F}^{\mathbf{k}}$ partition $\mathbf{X}$ completely into $|\mathbf{X}|/k$ disjoint partitions. Then, $\Gamma' : \mathbf{G}^{\mathbf{Y}} \rightarrow \mathbf{F}$, the transform regime-dual of Γ, is given by

$$\Gamma'(\mathbf{a}_c) = \underset{\mathbf{y} \in \mathbf{Y}}{\Gamma} \left(g^{-1}(\mathbf{a}_c(\mathbf{y})) \right). \tag{124}$$

The use of a transform-regime dual, instead of an analogue, is appropriate here, since Γ returns one value, rather than an image on $\mathbf{X}$. In practice, the output of Γ may not be in $\mathbf{G}$ if $|\mathbf{X}|$ or k is large, since $\mathbf{G}$ is designed to represent single pixel values, whereas $\mathbf{F}$ (by definition of Γ) contains values in *range*(Γ).

3.4.8.3. Example. Under constraint of the preceding technique, if $\Gamma = \Sigma$ and $g = \Sigma$ (i.e., T_4 denotes block summation), then g^{-1} does not exist, but $\Gamma' = \Sigma$ (where summation is applied to block sums in *range*(T_4) that are produced by g), since addition is associative.

We next discuss the more challenging task of deriving analogues over *range*(T_4) of the pointwise image operations. For purposes of simplicity, we begin with the transform-regime dual.

3.4.8.4. Technique: Dual pointwise operations. Let ${}^{M}T_4$ be specified in fine-granular form as the scalar operation $g : \mathbf{F}^{\mathbf{k}} \rightarrow \mathbf{G}$. Assume that there exists an inverse operation $g^{-1} : \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{k}}$ or an ϵ-near approximation to the inverse denoted by $g^{*} : \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{k}}$, and let $h^{*} : \mathbf{Y} \rightarrow 2^{\mathbf{X}}$ denote an indexing function that accepts a point $\mathbf{y} \in \mathbf{Y}$ and returns the domain $\mathrm{D} \subset \mathbf{X}$ of the $\mathbf{y}$-th encoding block. Let $f : \mathbf{F}^{\mathbf{k}} \times 2^{\mathbf{X}} \rightarrow 2^{\mathbf{F}^{\mathbf{X}}}$ accept a tuple from the output of g^{-1} and a subset of $\mathbf{X}$ from the output of h^{*}, and return the subimage on $\mathbf{F}^{\mathbf{X}}$

whose domain is specified by the output of h^*. Given source images $\mathbf{a},\mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ that are operated upon by $\bigcirc : \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$, we outline the derivation of $\Omega : \mathbf{G}^{\mathbf{Y}_1} \times \mathbf{G}^{\mathbf{Y}_2} \rightarrow \mathbf{F}^{\mathbf{X}}$, where $\mathbf{Y}_1, \mathbf{Y}_2 \subset \mathbf{Y}$, which is the dual of $\bigcirc$ over *range*(T_4), as

Step 1. Assume that $\mathbf{a}_c \in \mathbf{G}^{\mathbf{Y}_1}$ and $\mathbf{b}_c \in \mathbf{G}^{\mathbf{Y}_2}$ are computed as $\mathbf{a}_c = T_4(\mathbf{a})$ and $\mathbf{b}_c = T_4(\mathbf{b})$.

Step 2. Then, $\mathbf{c} = \mathbf{a} \bigcirc \mathbf{b}$ is computed with the dual operation Ω as

$$\begin{aligned} \mathbf{c} &= \Omega(\mathbf{a}_c,\mathbf{b}_c) \\ &= \bigcup_{\mathbf{y}\in\mathbf{Y}_1} \bigcup_{\mathbf{z}\in\mathbf{Y}_2} f\left(g^{-1}[\mathbf{a}_c(\mathbf{y})],\, h^*(\mathbf{y})\right)|_{h^*(\mathbf{y})\cap h^*(\mathbf{z})} \bigcirc f\left(g^{-1}[\mathbf{a}_c(\mathbf{z})],\, h^*(\mathbf{z})\right)|_{h^*(\mathbf{y})\cap h^*(\mathbf{z})} . \end{aligned} \tag{125}$$

Thus, $\mathbf{c} = \mathbf{a} \bigcirc \mathbf{b} = \mathbf{a}_c \,\Omega\, \mathbf{b}_c$.

The double union operation in Equation 125 is important implementationally, as discussed in Section 3.4.8.6. We briefly digress to consider an alternative formulation.

3.4.8.5. Technique: Analogous pointwise operations. Given f and h^*, assume that there exists a transform $V_4 : 2^{\mathbf{F}^{\mathbf{X}}} \rightarrow \mathbf{G}$ that accepts the output of f and produces a value in *range*(*range*(T_4)). Further assume that a tuple constructor Υ can accumulate values in $\mathbf{G}$ to form a transformed image in $\mathbf{G}^{\mathbf{Y}}$. Under constraint of the technique shown in Section 3.4.8.4, and given V_4 and Υ, a dual operation Ω (reference Equation 125) can be converted to an analogue $\bigcirc' : \mathbf{G}^{\mathbf{Y}_1} \times \mathbf{G}^{\mathbf{Y}_2} \rightarrow \mathbf{G}^{\mathbf{Y}_3}$ of $\bigcirc$ over *range*(T_4), where $\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3 \subset \mathbf{Y}$. The analogous operation is specified as

$$\begin{aligned} \mathbf{c}_c &= \bigcirc'(\mathbf{a}_c,\mathbf{b}_c) \\ &= \Upsilon_{\mathbf{y}\in\mathbf{Y}_1,\mathbf{z}\in\mathbf{Y}_2} V_4\left[f\left(g^{-1}[\mathbf{a}_c(\mathbf{y})],\, h^*(\mathbf{y})\right)|_{h^*(\mathbf{y})\cap h^*(\mathbf{z})} \bigcirc f\left(g^{-1}[\mathbf{a}_c(\mathbf{z})],\, h^*(\mathbf{z})\right)|_{h^*(\mathbf{y})\cap h^*(\mathbf{z})}\right], \end{aligned} \tag{126}$$

where $\mathbf{c}_c$ equals or approximates $T(\bigcirc(\mathbf{a},\mathbf{b}))$. Here, the tuple constructor is employed only once, versus the double union operations in Equation 125, to avoid constructing a tuple of tuples. If we employed double constructors Υ, then the resultant image would have two levels of structure. That is, each high-level tuple would contain a list that would contain one

or more lists of pixels in the transformed image $\mathbf{c}_c$. This image would then be tuple-valued, instead of G-valued.

3.4.8.6. Remark. The preceding techniques yield two important observations about the behavior of dual and analogous pointwise operations over $range(T_4)$, namely,

1. The double union of values under restriction of the block intersection $h^*(\mathbf{y}) \cap h^*(\mathbf{z})$ (per Equation 125) implies that dual pointwise operations over subsets of $\mathbf{Y}$ that are not identically partitioned blockwise can incur work of $\mathbf{O}(|\mathbf{Y}|^2)$ block intersections.
2. If the source images $\mathbf{a}$ and $\mathbf{b}$ are partitioned differently, then the block intersection operation can lead to fragmentation of blocks encoded in $\mathbf{a}_c$ and $\mathbf{b}_c$. Such fragmentation can also occur in analogous pointwise operations, as described by the restriction in Equation 126. In Figure 3, we portray fragmentation in the compressed image $\mathbf{c}_c = \mathbf{a}_c \bigcirc' \mathbf{b}_c$ that would be produced by an analogous pointwise operation. Fragmentation can increase the space complexity of $\mathbf{c}_c$ and thus decrease CR_d. This adversely affects the compression ratio and decreases computational efficiencies that depend upon CR_d. In a cascade of such compressive operations, it is possible that CR could become sub-unitary, thus obviating the chief goal of compressive processing, namely, reduced storage and computational costs.

In order to minimize intersection overhead and block fragmentation, the images $\mathbf{a}$ and $\mathbf{b}$ should be identically partitioned. If identical partitioning is employed, then the following simplification holds.

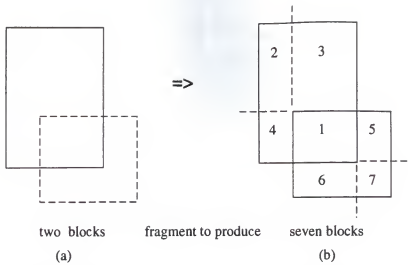


Figure 3. Example of block fragmentation in pointwise compressive operations over non-identically tesselated domains: (a) representation of the **y**-th block in **a** (solid line) and **b** (dotted line), (b) resultant block decimation required to process the compressed images $\mathbf{c}_c = \mathbf{a}_c \bigcirc' \mathbf{b}_c$ such that corresponding sub-blocks are amenable to pointwise combination.

3.4.8.7. Technique: Analogous pointwise operations with corresponding blocks. Given source images $\mathbf{a},\mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ that are operated upon by $\bigcirc : \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ and $T_4 : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ (implemented in terms of a block encoding $g : \mathbf{F}^{\mathbf{k}} \rightarrow \mathbf{G}$, we derive $\bigcirc' : \mathbf{G}^{\mathbf{Y}} \times \mathbf{G}^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, where $\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3 \subset \mathbf{Y}$, an analogue of $\bigcirc$ over *range*(T_4), as follows:

Step 1. Assume that $\mathbf{a}_c \in \mathbf{G}^{\mathbf{Y}}$ and $\mathbf{b}_c \in \mathbf{G}^{\mathbf{Y}_2}$ denote the compressed images $\mathbf{a}_c = T_4(\mathbf{a})$ and $\mathbf{b}_c = T_4(\mathbf{b})$.

Step 2. Assuming that g has an inverse $g^{-1} : \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{k}}$, the compressed image $\mathbf{c}_c = T_4(\mathbf{a} \bigcirc \mathbf{b})$ is expressed in terms of the analogous operation $\bigcirc$, as

$$\mathbf{c}_c = \bigcirc'(\mathbf{a}_c, \mathbf{b}_c) = \bigcup_{\mathbf{y} \in \mathbf{Y}} g^{-1}\left(\bigcirc\left(g^{-1}[\mathbf{a}_c(\mathbf{y})], g^{-1}[\mathbf{b}_c(\mathbf{y})]\right)\right). \tag{127}$$

If $\bigcirc'$ was an ϵ-near analogue of $\bigcirc$, then the equality in the preceding expression would be replaced by an approximation relation, and g would have an approximate inverse $g^* : \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{k}}$.

3.4.8.8. Remark. Although image-template operations can be derived via composition of the global reduce and pointwise operations, such formulations may not be tractable analytically. For example, one could incur high computational cost attempting to compute an image-template operation couched in terms of Equation 126, due to the block intersection operation. Thus, we next discuss image-template operations in terms of manipulating the substitutions inherent in the transform classes 4–A and 4–B.

3.4.8.9. Observation. Let the Class 4 transform $T_4: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, expressed at medium granularity as ${}^{\mathrm{M}}T_4: (\mathbf{F}^{\mathbf{k}})^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, be a homomorphism that is implemented at fine granularity in terms of a substitution ${}^{\mathrm{F}}T_4 = g: \mathbf{F}^{\mathbf{k}} \rightarrow \mathbf{G}$. If the source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, and the indexing function $h^*: \mathbf{Y} \rightarrow 2^{\mathbf{X}}$, then the encoded image $\mathbf{a}_c$ is given by

$$\mathbf{a}_c = T_4(\mathbf{a}) = \bigcup_{\mathbf{y} \in \mathbf{Y}} \left(\mathbf{y}, g\left(\mathbf{a}|_{h^*(\mathbf{y})}\right)\right). \tag{128}$$

Here, the equivalence relation is used, since the right member of Equation 126 denotes a set that contains the graph of the compressed image $\mathbf{a}_c$. Let the template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, and assume that the image-template operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is applied to yield $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{t}) = \mathbf{a} \textcircled{\scriptsize T} \mathbf{t}$.

An analogue of $\bigcirc$ can be formulated in one of two ways:

Method 1. If $\bigcirc': \mathbf{G}^{\mathbf{Y}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, then from the definition of homomorphism, $T_4(\hat{\mathbf{a}}) = \bigcirc'(\mathbf{a}_c,\mathbf{t})$. Thus, $\mathbf{a}_c$ is combined with $\mathbf{t}$. However, $\bigcirc'$ must implement at least the maps $g^{-1}: \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{k}}$ or $g^*: \mathbf{G} \rightarrow \mathbf{F}^{\mathbf{k}}$ as well as h^*, in order to respectively convert values in $\mathbf{G}$ to a block of k values in $\mathbf{F}$, and points in $\mathbf{Y}$ to a subset of $\mathbf{X}$.

Method 2. Alternatively, if $\bigcirc': \mathbf{G}^{\mathbf{Y}} \times (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, then one may define a template $\mathbf{s} \in (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}$, such that $T_4(\hat{\mathbf{a}}) = \bigcirc'(\mathbf{a}_c,\mathbf{s}) = \mathbf{a}_c \textcircled{\scriptsize T}'\mathbf{s}$. If the basis operations $\bigcirc',\gamma': \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$ of $\textcircled{\scriptsize T}'$ differ from the basis operations $\bigcirc,\gamma: \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$ of $\textcircled{\scriptsize T}$, then it is possible for $\textcircled{\scriptsize T}'$ to differ from $\textcircled{\scriptsize T}$, the operation upon which $\bigcirc$ is based.

The formulation of $\bigcirc'$ shown in Method 1 can be readily implemented if g^{-1} or g^* and h are known. In practice, templates such as $\mathbf{s}$ may be easily specified at a high level, but may not be amenable to efficient implementation via an analogous image-template operation $\textcircled{T}'$. We briefly discuss such issues in the subsequent technique.

3.4.8.10. Technique: Special case of image-template operations. Assume that the template $\mathbf{s} \in (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}$ is applied to the compressed image $\mathbf{a}_c \in \mathbf{G}^{\mathbf{Y}}$. From the previous discussion of transform T_4, we know that each point $\mathbf{y} \in \mathbf{Y}$ corresponds to an encoding block domain $D \subset \mathbf{X}$. Considering the usual case where h^* is a bijection, we see that the image operation $\bigcirc(\mathbf{a},\mathbf{t}) = \mathbf{a} \textcircled{T} \mathbf{t}$ is space-variant, such that $\mathcal{S}(\mathbf{t}_u) = h(\mathbf{y})$, $\mathbf{u} \in \mathbf{X}, \forall \mathbf{y} \in \mathbf{Y}$, and the values in $\mathbf{a}|_{h^*(\mathbf{y})}$ are preserved in $\mathbf{a}_c(\mathbf{y})$, $\forall \mathbf{y} \in \mathbf{Y}$.

Since $\textcircled{T}$ is implemented in terms of the associative, commutative operations $\bigcirc, \gamma : \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$, the template $\mathbf{s}$ is defined by its weights as

$$\mathbf{s}_{\mathbf{y}}(\mathbf{v}) = \begin{cases} \mathrm{e} \text{ if } \mathbf{v} = \mathbf{y} \\ \mathrm{h} \text{ otherwise} \end{cases}, \ \mathbf{y} \in \mathbf{Y}, \tag{129}$$

where e denotes the identity of $\bigcirc$, and h denotes a value that is the zero of $\bigcirc$ as well as the identity of γ, if h exists.

Since composition is preserved under homomorphism, an analogue $\textcircled{T}' : \mathbf{G}^{\mathbf{Y}} \times (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, of $\textcircled{T}$ over *range*(T), can be defined in terms of $\bigcirc'$ and γ' in a manner that is symmetric to the definition of $\textcircled{T}$, i.e.,

$$(\mathbf{a}_c \textcircled{T}' \mathbf{s})(\mathbf{v}) = \mathop{\Gamma'}_{\mathbf{y} \in S(\mathbf{t}_v)} \mathbf{a}_c(\mathbf{y}) \circ' \mathbf{s}_{\mathbf{v}}(\mathbf{y}), \ \mathbf{v} \in \mathbf{Y}. \tag{130}$$

From merely $\textcircled{T}'$, we compute $\bigcirc'(\mathbf{a}_c, \mathbf{s}) = \mathbf{a}_c \textcircled{T}' \mathbf{s}$.

3.4.8.11. Observation. The preceding case can be highly restrictive in practice, for the following reasons. First, the block encoding kernel $g: \mathsf{F}^{\mathbf{k}} \rightarrow \mathsf{G}$ is a bijection in the case of lossless encoding only. However, most Class 4 transforms implement inherently lossy forms of block encoding (e.g., VQ and JPEG). In such cases, T_4 (and, therefore, the basis function g) would have an approximate inverse only. Thus, $\bigcirc'$, as stated in Equation 130, might not exist for lossy encoding since T_4 might not be a homomorphism. Instead we would treat T_4 as having an ϵ-near approximation T_4^* to its inverse.

Second, even if we have a lossless version of T_4, if analogues of $\bigcirc$ and γ over $range(T_4)$ are unknown or intractable, then $\textcircled{T}'$ (and, therefore, $\bigcirc'$) cannot be derived as shown in the preceding section. Instead, approximations to $\bigcirc'$ and γ' must be derived, then composed to form $\textcircled{T}'$.

Third, even if T_4 is lossless and analogues of $\bigcirc$ and γ exist, the formulation of the compressive template $\mathbf{s}$ may be difficult due to lossy transformation of the boundary descriptions of adjacent encoding blocks. Thus, we prefer to consider each transform in Class 4 as a special case of the image-template formulations given previously.

3.4.8.12. Remark. In practice, it is often useful to approximate image-template operations by exploiting some feature of the source image that is preserved in the compressed image. For example, if T preserves boundary information in the compressed image, then such information may be employed by a compressive edge detection operation that filters out non-boundary information. We further illustrate this process in Chapter 9.

We next discuss the complexity of operations over transforms in Classes 1–4, and present a high-level model for compressive computation.

CHAPTER 4
COMPRESSIVE PROCESSING — COMPUTATIONAL COMPLEXITY AND DATA SECURITY

In Chapter 1, we stated that computational efficiency is a key premise of processing compressed data, due to the reduced number of operations that may be required by fewer input data. The realization of such efficiency assumes that the transform properties problem can be solved to yield one or more transform-regime analogues of image operations. Implementationally, we assume that such analogues exhibit computational cost less than or equal to that incurred by the corresponding image operations. However, such assumptions may not hold universally in practice.

In the previous chapter, we proposed high-level techniques for the determination of analogous operations. In this chapter, we present theory that supports implementational analysis of compressive processing. We first discuss the complexity of image algebra operations, then derive expressions for the complexity of analogous operations over transformed imagery (Section 4.1). We next examine the complexity of computations over the range space of generalized transforms in Classes 1–4 of our transform taxonomy (Section 4.2). The resulting theory is used to describe preliminary optimization procedures for compressive computational systems. A discussion of information theory is given in Section 4.3, which leads naturally to a consideration of data security. Similar to the assertion of computational speedup due to compressive processing, we state that the data security of computational operations can be increased by processing encrypted data. Thus, Section 4.4 overviews concepts and theory of cryptography and cryptanalysis, while Section 4.5 discusses theory and feasibility of encryptive processing.

4.1. Complexity of Image Algebra Operations.

In Chapter 2, we state that image algebra consists of pointwise, global-reduce, and image-template operations [54,55]. In the latter category are included the image-image convolutions, as well as operations between templates. In this section, we analyze the work required to compute nonrecursive image algebra operations in an implementational context, beginning with pointwise operations.

4.1.1. Complexity of Pointwise Image Operations.

4.1.1.1. Lemma. Each computation of the nonconstant function $\bigcirc : \mathbf{F}^m \rightarrow \mathbf{F}^k$ incurs minimum work of

(i) one input operation,

(ii) one computation, and

(iii) one output operation.

Proof. The proof follows trivially from the assumption that each tuple that comprises an element of $\mathbf{F}^m$ or $\mathbf{F}^k$ can be encoded as one storage element whose input or output requires one pair of I/O operations (read and write). □

Since the pointwise operation $\bigcirc : (\mathbf{F}^{\mathbf{X}})^m \rightarrow \mathbf{F}^{\mathbf{X}}$ is induced over $\mathbf{X}$ by the scalar function $\bigcirc : \mathbf{F}^m \rightarrow \mathbf{F}$, one can use the preceding lemma to predict the work required by $\bigcirc$.

4.1.1.2. Example. Denote $\mathbb{B} = \{0, 1\}$, for purposes of brevity. A Boolean logic operation $\bigcirc : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ over the image $\mathbf{a} \in \mathbb{B}^{\mathbf{X}}$ induces the function $\bigcirc : \mathbb{B}^{\mathbf{X}} \times \mathbb{B}^{\mathbf{X}} \rightarrow \mathbb{B}^{\mathbf{X}}$, which requires $|\mathbf{X}|$ invocations of $\bigcirc$. Thus, if an element in *domain*($\bigcirc$) is encoded as a two-bit integer in $\mathbb{B}^2$, or as a one-bit encoding of a signed integer, then $2 \cdot |\mathbf{X}|$ I/O operations will be required. In other words, one operation is required to input an element in $\mathbb{B}^2$, and another for output. Otherwise, the I/O overhead would increase to $3 \cdot |\mathbf{X}|$ operations.

4.1.2. Complexity of Global Reduce Operations.

4.1.2.1. Lemma. If $\Gamma: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}$ is a global-reduce operation that is based upon the associative, commutative function $\gamma: \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$, then one computation of Γ requires minimum work of

(i) $|\mathbf{X}|$ input operations,

(ii) $|\mathbf{X}| - 1$ computations of γ, and

(iii) one output operation.

Proof. The proof follows trivially from Lemma 4.1.1.1 and the assumption that the computational cost includes I/O overhead. □

4.1.2.2. Example. Let $\mathbf{X}$ be an m×n-pixel array and source image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, and let the associative, commutative function $\gamma = \vee$. The corresponding global reduce function $\Gamma(\mathbf{a}) = \vee\mathbf{a}$ requires work of at least $|\mathbf{X}| + 1$ I/O operations and $|\mathbf{X}| - 1$ comparisons.

4.1.3. Complexity of Image-template Operations.

4.1.3.1. Lemma. If image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$, then the right product $\mathbf{b} = \mathbf{a} \textcircled{\gamma} \mathbf{t} \in \mathbf{F}^{\mathbf{Y}}$ exhibits the following minimum work requirement:

(i) N_i, the number of input operations, and N_c, the number of computations of $\circ: \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$, are bounded as

$$|\mathbf{Y}| \leq N_i, N_c \leq \sum_{\mathbf{y} \in \mathbf{Y}} |\mathcal{S}(\mathbf{t}_\mathbf{y})|, \tag{131}$$

(ii) the number of computations of $\gamma: \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$ is bounded as

$$1 \leq N_\gamma \leq \sum_{\mathbf{y} \in \mathbf{Y}} (|\mathcal{S}(\mathbf{t}_\mathbf{y})| - 1), \tag{132}$$

(iii) the number of output operations is bounded as

$$1 \leq N_o \leq |\mathbf{Y}|. \tag{133}$$

Proof. The proof is outlined as follows:

(i) N_i follows from the definition of template support, and the fact that a minimum of $|\mathbf{Y}|$ inputs must be examined by $\mathbf{t}$. N_c follows from N_i, the definition of $\textcircled{T}$ and Lemma 4.1.1.1.

(ii) N_γ follows from N_i, the definition of $\textcircled{T}$, Lemma 4.1.1.1, and Example 4.1.2.2.

(iii) The lower bound on N_o follows from the implicit assumption that the output image $\mathbf{b}$ is initialized to an image $\mathbf{b}_0$, and that it is possible for $(\mathbf{a}\,\textcircled{T}\mathbf{t})(\mathbf{y}) = \mathbf{b}_0(\mathbf{y})$, $\mathbf{y} \in \mathbf{Y}$, which would obviate the necessity of output to $\mathbf{b}(\mathbf{y})$. If all values in $\mathbf{b}$ were replaced by different values, then $|\mathbf{Y}|$ write operations would be required. □

The image-template left product presents a symmetric case and operations between templates are similarly analyzed.

4.1.4. Complexity of Operations over Transformed Data

In this section, we derive expressions for the complexity of computations over the range space of a generalized image transform T: $\mathsf{F}^{\mathbf{X}} \rightarrow \mathsf{G}^{\mathbf{Y}}$. We mirror the preceding development by discussing pointwise, global-reduce, and image-template operations.

4.1.4.1. Theorem. Given a homomorphic computational system $X_H = (\{\mathsf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathsf{G}^{\mathbf{Y}}, \bigcirc'\})$ with pointwise operation $\bigcirc$, the computation of a pointwise analogue $\bigcirc'$ requires work that is bounded as

(i) $|\mathbf{Y}| \leq N_i \leq 2 \cdot |\mathbf{Y}|$ input operations,

(ii) $1 \leq N_c \leq |\mathbf{Y}|$ computations of the scalar operation $\bigcirc'$, and

(iii) $N_o \geq |\mathbf{Y}|$ output operations.

Proof. Statements i) through ii) follow from Lemma 4.1.1.1 and the definition of a transform-regime analogue. □

4.1.4.2. Remark. It follows from the preceding theorem that the computation of pointwise operations over *range*(T) could be achieved in $\mathbf{O}(|\mathbf{Y}|)$ time, as stated in the introduction. However, if domain fragmentation occurs (for example, due to unequal tesselation of the inputs of $\bigcirc'$) the domain of the output of $\bigcirc'$ may be larger than $|\mathbf{Y}|$. That is why we state that $N_o \geq |\mathbf{Y}|$ in Theorem 4.1.4.1(iii). Alternatively, $\mathbf{Y}$ may be preserved by $\bigcirc'$, but the output of $\bigcirc'$ may have multiple values per point in $\mathbf{Y}$ that cannot be written in one I/O operation.

4.1.4.3. Theorem. Given T: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, let the global reduce operation $\mathbf{\Gamma}$: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}$ have an analogue $\mathbf{\Gamma}'$: $\mathbf{G}^{\mathbf{Y}} \rightarrow \mathbf{G}$ over *range*(T). If $\mathbf{\Gamma}$ is implemented in terms of an associative; commutative function γ: $\mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$ that has an analogue γ': $\mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$, then the computation of $\mathbf{\Gamma}'$ requires work that is bounded as

(i) $|\mathbf{Y}|$ input operations,

(ii) $N_c \geq |\mathbf{Y}| - 1$ computations of γ', and

(iii) $N_o \geq 1$ output operations.

Proof. Statements i) through iii) follow directly from Lemma 4.1.1.1 and the givens. Note that the bound on N_c is constrained by the implicit assumption that γ' is a pointwise analogue of γ. □

4.1.4.4. Remark. Consider the implications of the preceding theorems for parallel computation. Referring to Theorem 4.1.4.1, assume that (a) the analogue $\bigcirc'$ is a pointwise operation that requires $|\mathbf{Y}| < |\mathbf{X}|$ computations of the scalar operation $\bigcirc'$: $\mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$, (b) that $\bigcirc'$ is computed with degree of parallelism N, and (c) that the speed of computation is invariant to the ratio $r = siz(\mathbf{F})/siz(\mathbf{G})$ for purposes of convenience. In such cases, the compression ratio reduces to $CR = r \cdot CR_d = r \cdot |\mathbf{X}|/|\mathbf{Y}|$. Neglecting I/O cost and

interprocessor communication overhead, it is easily verified that the speedup achievable via compressive parallel processing, as opposed to noncompressive sequential processing, is given by $\eta_{(\mathbf{O}',N)} \propto N \cdot CR_d$. In practice, the proportionality constant would likely be nonlinear in N due to the usual communications and control overhead. Thus, unless otherwise stated, we assume that computational cost predominates.

Per Lemma 4.1.2.1, assume that the global reduce operation $\boldsymbol{\Gamma}$ requires $|\mathbf{X}| - 1$ invocations of its basis function γ. With sufficient parallelism and exclusive of I/O and control overhead, it is possible to compute γ in $\lceil \log_2|\mathbf{X}| \rceil \cdot \Delta t_\gamma$ time, where Δt_γ denotes the computational delay incurred by γ. Likewise, if $\boldsymbol{\Gamma}'$, the analogue of $\boldsymbol{\Gamma}$ over *range*(T), requires $|\mathbf{Y}| - 1$ invocations of the function γ', then with sufficient parallelism, it is possible to compute γ' in $\lceil \log_2|\mathbf{Y}-1| \rceil \cdot \Delta t_{\gamma'}$ time, exclusive of I/O and control overhead. Letting $T_{(x,y)}(n)$ denote the time complexity of operation x on an architecture of y processors with n input data, the computational speedup associated with $\boldsymbol{\Gamma}'$ (versus $\boldsymbol{\Gamma}$) would be given by

$$\eta_{(\boldsymbol{\Gamma},\boldsymbol{\Gamma}',|\mathbf{X}|)} = \frac{T_{(\boldsymbol{\Gamma},|\mathbf{X}|/2)}(|\mathbf{X}|)}{T_{(\boldsymbol{\Gamma}',|\mathbf{Y}|/2)}(|\mathbf{Y}|)} = \frac{\lceil \log|\mathbf{X}| \rceil}{\lceil \log|\mathbf{Y}| \rceil}, \tag{134}$$

4.1.4.5. Remark. Initially, the preceding equation may not appear to yield a remarkable increase in processing efficiency. For example, rather than exhibiting the parallel efficiency of $\mathbf{O}(CR_d)$ that is characteristic of analogous pointwise operations, the analogous global reduce operations exhibit a parallel efficiency of $\mathbf{O}(\log|\mathbf{X}|/\log|\mathbf{Y}|)$, which is less than $\log(CR_d) = \log(|\mathbf{X}|) - \log(|\mathbf{Y}|)$. For example, if $|\mathbf{X}| = 1M$, and $|\mathbf{Y}| = 256K$, then $CR_d = 4$, and $\log|\mathbf{X}|/\log|\mathbf{Y}| = 10/8 = 1.25$. In contrast, $\log(CR_d) = 2$.

However, this apparently low efficiency is compensated for by the decreased *cost* of processing, which is computed as $C = N \cdot T$, where T denotes time complexity. In the second member of Equation 134, note that $\boldsymbol{\Gamma}$ requires $|\mathbf{X}|/2$ processors to achieve logarithmic computation time, while $|\mathbf{Y}|/2$ processors are required for computing $\boldsymbol{\Gamma}'$ under the same constraint. Thus, the total processing efficiency is given by

$$\eta^{cost}_{(\boldsymbol{\Gamma},\boldsymbol{\Gamma}',|\mathbf{X}|)} = \frac{|\mathbf{X}| \cdot T_{(\boldsymbol{\Gamma},|\mathbf{X}|/2)}(|\mathbf{X}|)}{|\mathbf{Y}| \cdot T_{(\boldsymbol{\Gamma}',|\mathbf{Y}|/2)}(|\mathbf{Y}|)} = CR_d \cdot \eta_{(\boldsymbol{\Gamma},\boldsymbol{\Gamma}',|\mathbf{X}|)}. \tag{135}$$

Thus, the analogous global reduce function can produce a computational efficiency proportional to the domain compression ratio.

We next consider computation of $\textcircled{?}'$, the analogue of the image-template operation $\textcircled{?}$ over *range*(T). For purposes of simplicity, we assume that there exists a space-invariant template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ for which an analogous template $\mathbf{s} \in (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}$ exists, per the discussion in Section 3.2.2 of HCS and ICS that embody image-template operations.

4.1.4.6. Theorem. Let a source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ be convolved with template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ to yield the operation $\bigcirc(\mathbf{a},\mathbf{t}) = \mathbf{a} \textcircled{?} \mathbf{t}$, where $\textcircled{?}$ is comprised of operations $\circ, \gamma : \mathbf{F} \times \mathbf{F} \to \mathbf{F}$. Assume that T: $\mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{Y}}$ is a homomorphism, and that $\mathbf{a}_c = T(\mathbf{a})$. Additionally, let template $\mathbf{s} \in (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}}$, and let $\bigcirc = \textcircled{?}$ have an analogue $\textcircled{?}'$ over *range*(T), such that $\bigcirc'(\mathbf{a}_c,\mathbf{s}) = \mathbf{a}_c \textcircled{?}' \mathbf{s} = T(\mathbf{a} \textcircled{?} \mathbf{t})$, where $\textcircled{?}'$ is comprised of operations $\circ', \gamma' : \mathbf{F} \times \mathbf{F} \to \mathbf{F}$ that are the analogues of $\circ$ and γ. The computation of $\bigcirc'$ requires work that is bounded as

(i) $|\mathbf{Y}| \leq N_i \leq \sum_{\mathbf{y} \in \mathbf{Y}} |\mathcal{S}(\mathbf{s}_\mathbf{y})|$ input operations,

(ii) $|\mathbf{Y}| \leq N_c \leq \sum_{\mathbf{y} \in \mathbf{Y}} |\mathcal{S}(\mathbf{s}_\mathbf{y})|$ computations of $\circ'$

and $|\mathbf{Y}| - 1 \leq N_c \leq \sum_{\mathbf{y} \in \mathbf{Y}} |\mathcal{S}(\mathbf{s}_\mathbf{y})| - (|\mathbf{Y}|+1)$ computations of γ', as well as

(iii) $1 \leq N_o \leq |\mathbf{Y}|$ output operations.

Proof. Statements i) through iii) follow directly from Lemmas 4.1.4.1 and 4.1.4.3. □

When $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ is space-invariant, the left product is identically characterized.

4.1.4.7. Remark. As we showed for pointwise and global reduce operations, an analogue to the image-template product can, in principle, be computed in $\mathbf{O}(|\mathbf{Y}|)$ work. However, a more realistic estimate is $\mathbf{O}\left(|\mathbf{Y}| \cdot \sum_{\mathbf{y} \in \mathbf{Y}} |\mathcal{S}(\mathbf{s}_\mathbf{y})|\right)$ work. The latter estimate is generally bounded above by $\mathbf{O}(|\mathbf{Y}|^3)$, since the usual upper bound of $\mathbf{O}(|\mathbf{Y}|^2)$ associated with convolution is multiplied by a maximum of $\mathbf{O}(|\mathbf{Y}|)$ search operations that could be incurred in locating the point $\mathbf{y} \in \mathbf{Y}$ corresponding to a given source point $\mathbf{x} \in \mathbf{X}$. Such conditions hold especially if the values of $\mathbf{a}_c$ encode one or more points in $\mathbf{X}$ that cannot be predicted

from $\mathbf{y}$ using one invocation of an indexing function. In Chapter 11, we discuss this problem as it pertains to the special case of SIMD parallel compressive computation and show how certain instances of this problem can be solved implementationally via lookup tables.

4.1.4.8. Summary. We have seen that $|\mathbf{Y}|$, the number of pixels in the encoded image, is the order of work required by compressive pointwise analogues to pointwise image operations. When such analogues are composed to yield an analog to the image-template product, a speedup of $\mathbf{O}(\mathrm{CR}_d)$ can be achieved under two constraints. First, the compressed domain must not be enlarged due to fragmentation by the analogous operations. Second, the work required by an analogous operation with respect to the corresponding image operation should be a linear function of the operand size at worst, and a constant function at best.

Given the fact that $|\mathbf{Y}|$ is a key parameter of the compression ratio CR, it is reasonable to expect that the work required for computation of a given analogous operation can be predicted in terms of CR. Thus, in the following section, we express the computational speedup in terms of a limit called the *critical compression ratio*, above which supra-unitary computational speedup occurs.

4.2. Complexity of Compressive Computation.

In Chapter 3, we defined a compressive transform and its compression ratio. In this section, we first elaborate the efficiency η due to computation over the range space of a transform T. Second, we derive the *critical compression ratio* CR_c. When the compression ratio $\mathrm{CR} > \mathrm{CR}_c$, then $\eta > 1$, i.e., a computational speedup occurs with respect to noncompressive processing.

4.2.1. Basic Theory.

Per the discussion of sections 4.1.4.4 and 4.1.4.5, we define the concept of computational speedup in a compressive homomorphic computational system, then relate the speedup to the compression ratio.

4.2.1.1. Definition. Given a homomorphic computational system $X_H = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \bigcirc'\})$, the *computational efficiency* η due to processing on an N-processor architecture A with operation $\bigcirc'$ (an analogue over *range*(T) of an image operation $\bigcirc$) is given in general form as

$$\eta(X_H) \equiv \eta_{(\bigcirc,\bigcirc',N)} = \frac{W_{A(\bigcirc,N)}(|\mathbf{X}|)}{W_{A(\bigcirc,N)}(|\mathbf{Y}|)}. \tag{136}$$

For purposes of simplicity, we generally assume sequential processing unless otherwise stated and omit the subscripts of η as well as A and N, which can be inferred from context.

We next present an assumption that simplifies the relationship between η and CR.

4.2.1.2. Assumption. Let homomorphism $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, image operation $\bigcirc: (\mathbf{F}^{\mathbf{X}})^m \rightarrow \mathbf{F}^{\mathbf{X}}$, and let $\bigcirc': (\mathbf{G}^{\mathbf{Y}})^m \rightarrow \mathbf{G}^{\mathbf{Y}}$ denote the analogue of $\bigcirc$ over *range*(T).

(i) Assume that the work $W_{\bigcirc,N}(n)$ can be expressed as a tractable, continuous function that has an inverse denoted by $W^{-1}_{\bigcirc,N}(n)$.

(ii) When work incurred by $\bigcirc$ is written as $W_{\bigcirc,N}(n)$ or is instantiated as the sequential time complexity $T_{seq(\bigcirc,N)}(|\mathbf{X}|) \equiv W_{(\bigcirc,1)}(|\mathbf{X}|)$, we assume that the effect of *siz*($\mathbf{F}$) upon W is accounted for in the formulation or expression of W, unless stated otherwise.

4.2.1.3. Remark. Assumption 4.2.1.2(i) is justified in practice, since continuous, invertible functions can generally be fitted to data that describe the performance of an operation on a specific architecture with a given number of inputs. It is beyond the scope of this chapter to discuss the complex implementational issues pertaining to the accuracy with which the complexity function W represents the performance of a given operation. Rather, we assume for purposes of convenience that the representation is accurate, and present implementational analyses in Parts II and III.

4.2.1.4. Lemma. Let a homomorphic computational system $X_H = (\{F^X, \bigcirc\}, \{T, \Theta\}, \{G^Y, \bigcirc'\})$ and let the ratio

$$r_{A,N}(\bigcirc, \bigcirc', n) = \frac{T_{\bigcirc,N}(n)}{T_{\bigcirc',N}(n)}. \tag{137}$$

Given the simplistic assumption that $W_{\bigcirc}(n) \propto W_{\bigcirc'}(n) \propto n$ for all $n \leq |\mathbf{X}|$, then the efficiency due to processing over the range space of a compressive transform T is given by

$$\eta(X_H) = \frac{W_{\bigcirc,N}(CR_d \cdot |\mathbf{Y}|)}{r_{A,N}(\bigcirc, \bigcirc', |\mathbf{Y}|) \cdot W_{\bigcirc,N}(|\mathbf{Y}|)}. \tag{138}$$

Proof. From the definition of computational speedup and the definition of an HCS, we have the more general expression that

$$\eta = \frac{W_{A(\bigcirc,N)}(|\mathbf{X}|)}{W_{A(\bigcirc',N)}(|\mathbf{Y}|)}. \tag{139}$$

If $W_{\bigcirc}(n) \propto W_{\bigcirc'}(n) \propto n$, then we can (a) separate the effects of domain cardinality and value set space requirement, (b) substitute $|\mathbf{X}| = |\mathbf{Y}| \cdot CR_d$, and (c) introduce r into the denominator, thus obtaining Equation 138. □

4.2.1.5. Remark. The assumption of linearity between $W_{\bigcirc}$ and $W_{\bigcirc'}$ is not fanciful, but holds frequently in practice. In particular, consider the case when the range spaces of T's input and output conform to the relationship $siz(\mathbf{F}) \propto siz(\mathbf{G})$, and observe that W is usually proportional to $siz(\mathbf{F})$ but not to $|\mathbf{F}|$.

Given the foregoing relationship between η and CR_d, we can derive the critical compression ratio CR_c. For purposes of simplicity, we continue to denote $W_{\bigcirc}(n) \equiv W_{A(\bigcirc,N)}(n)$ and denote $r(n) \equiv r_{(A,N)}(\bigcirc, \bigcirc', n)$ as well as $\eta \equiv \eta(X_H)$

4.2.1.6. Definition. Given a homomorphic computational system X_H, if the compression ratio $CR(T)$ is less than the *critical compression ratio* CR_c, then the efficiency $\eta(X_H) > 1$.

4.2.1.7. Theorem. If a homomorphic computational system $X_H = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \bigcirc'\})$ and $W_{\bigcirc}(n) \propto W_{\bigcirc'}(n) \propto n$, then the critical compression ratio is given by

$$CR_c = \frac{CR_r}{|\mathbf{Y}|} \cdot W_{\bigcirc}^{-1}(r(n) \cdot W_{\bigcirc}(|\mathbf{Y}|)). \tag{140}$$

Proof. Per Assumption 4.2.1.2, the complexity function $W_{\bigcirc}$ has an inverse $W_{\bigcirc}^{-1}$ that accepts a measure of time and returns the number of elements upon which $\bigcirc$ operates in the given time. Since $W_{\bigcirc}(n) \propto W_{\bigcirc'}(n) \propto n$, the effects of domain cardinality and *siz* of the value set are separable. Regrouping the terms of Equation 138, and applying the definitions of CR_d and CR_r, we apply $W_{\bigcirc}^{-1}$ to obtain the derivation

$$\begin{aligned} \eta \cdot r(n) \cdot W_{\bigcirc}(|\mathbf{Y}|) &= W_{\bigcirc}(CR \cdot |\mathbf{Y}|/CR_r) \\ W_{\bigcirc}^{-1}[\eta \cdot r(n) \cdot W_{\bigcirc}(|\mathbf{Y}|)] &= W_{\bigcirc}^{-1}[W_{\bigcirc}(CR \cdot |\mathbf{Y}|/CR_r)] \\ CR \cdot |\mathbf{Y}|/CR_r &= W_{\bigcirc}^{-1}[\eta \cdot r(n) \cdot W_{\bigcirc}(|\mathbf{Y}|)]. \end{aligned} \tag{141}$$

By constraining the preceding equation such that $\eta = 1$ when $CR = CR_c$, we can regroup terms to obtain Equation 140. If $CR > CR_c$, then $\eta > 1$. □

4.2.1.8. Remark. The preceding theorem is useful implementationally, since speedup and complexity as well as critical compression ratio can be predicted given T, $\bigcirc$, $\bigcirc'$, and knowledge of a given architecture A. Note that Equation 140 does not explicitly require knowledge of *domain*(T). As discussed in Section 4.4–4.5, data security may thus be enhanced due to the concealment of certain properties of *domain*(T). Inherent in the preceding statement is the tacit assumption that one would conceal from routine usage the details of how r, CR_d, or CR_r are computed.

In the following sections, we analyze the computational complexity of each transform class in a compressive computational scenario. It is shown that the complexity that results from processing over a compressive transform's range space is typically bounded below by $\mathbf{O}(|\mathbf{Y}|)$ work, which implies $\mathbf{O}(CR_d)$ efficiency. Since $CR \propto CR_d$, the computational speedup that results from compressive processing is claimed to be $\mathbf{O}(CR)$ in the typical case.

4.2.2. Complexity of Class 1 Compressive Processing

Class 1 presents a useful introductory case, since transforms in this class can be viewed as greylevel functions that may be one-to-one (pointwise bijections) or many-to-one (pointwise or neighborhood operations).

4.2.2.1. Lemma. Let $T_1: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ be a homomorphism and $\bigcirc: \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ be a pointwise image operation that has an analogue $\bigcirc': \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ over *range*(T_1). If $W_{\bigcirc}(n) \propto W_{\bigcirc'}(n) \propto n$, then the computational efficiency η incurred by the HCS $X_H = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \bigcirc'\})$ is given by

$$\eta(X_H) = \left(r(\bigcirc, \bigcirc', |\mathbf{X}|)\right)^{-1}, \tag{142}$$

where the inverse is multiplicative.

Proof. In Section 3.4.2, we showed that $CR(T_1) = 1$. By the definitions of η and r, we have

$$\eta = \frac{W_{\bigcirc}(|\mathbf{X}|)}{r(|\mathbf{X}|) \cdot W_{\bigcirc}(|\mathbf{X}|)} = r^{-1}(|\mathbf{X}|), \tag{143}$$

and the lemma holds. □

As a result, pointwise computation over *range*(T_1) derives its efficiency from the choice of $\bigcirc'$ versus $\bigcirc$. This effect is not due to data compression, which is nonexistent in Class 1 transforms. The following example illustrates this concept.

4.2.2.2. Example. If the Fourier transform $T_1 = \mathcal{F}: \mathbf{C}^{\mathbf{X}} \rightarrow \mathbf{C}^{\mathbf{X}}$, the transform-regime analogue of convolution $\bigcirc = \oplus$ of images or templates having circulant domains is given by $\bigcirc' = *$, which denotes Hadamard multiplication. Now, let $\mathbf{X}$ be an $m \times n$-pixel array. Note that $\oplus$ requires work $W_{\bigcirc}(m,n) = \mathbf{O}(m^2n^2)$ multiplications, while the work associated with Hadamard multiplication is given by $W_{\bigcirc'}(m,n) = \mathbf{O}(mn)$ multiplications. Neglecting the overhead due to Fourier transformation, the resultant efficiency is computed as

$$\eta = \frac{W_{\bigcirc}}{W_{\bigcirc'}} = \frac{\mathbf{O}(m^2n^2)}{\mathbf{O}(mn)} = \mathbf{O}(mn). \tag{144}$$

Thus, the effect $\eta > 1$ is due to the performance ratio $r(\bigcirc, \bigcirc', |\mathbf{X}|)$, as predicted by the preceding lemma.

However the analysis of efficiency is more complicated when the space requirement is introduced in addition to a computational speedup, as shown in the following example.

4.2.2.3. Example. Suppose that we have the computational system $X = (\{\mathbf{R}^{\mathbf{X}}, \oplus\}, \{\mathcal{F}, \Theta\}, \{\mathbf{C}^{\mathbf{X}}, *\})$, as in the previous example. Given $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ with circulant $\mathbf{X} \subset \mathbf{R}^n$, a spatially-invariant template $\mathbf{t} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$, and $|\mathbf{X}|$-fold parallelism, an image domain convolution $\hat{\mathbf{a}} = \mathbf{a} \oplus \mathbf{t}$ requires $|S(\mathbf{t}_y)| - 1$ I/O and addition operations on a SIMD-parallel mesh, where $y \in \mathbf{X}$. In contrast, given $\mathbf{a}_c = \mathcal{F}(\mathbf{a})$ and $\mathbf{s} = \mathcal{F}(\mathbf{t}_y)$, the

computation of $\hat{\mathbf{a}}_c = \mathbf{a}_c * \mathbf{s}$ requires one multiplication using $|\mathbf{X}|$ processors. The resulting efficiency is given in terms of computational *cost* by

$$\eta(X) = \frac{|\mathbf{X}| \cdot (|\mathcal{S}(\mathbf{t_y})| - 1) \cdot \Delta t_a}{|\mathbf{X}| \cdot \Delta t_m}, \tag{145}$$

where Δt_a and Δt_m denote the computational delay incurred by addition and multiplication. If $\mathbf{X}$ is an m×m-pixel array, then the preceding equation reduces to $\eta(X) = (|\mathcal{S}(\mathbf{t_y})| - 1) \cdot \Delta t_a / \Delta t_m$, which has the general form $k \cdot |\mathcal{S}(\mathbf{t_y})|$, where k denotes a constant. Thus, η is proportional to template size as well as the ratio of the time complexities of addition and multiplication operations.

4.2.2.4. Observation. It is easily verified that the preceding claims hold for unary and binary pointwise operations, as well as for global reduce operations that are constructed from binary pointwise operations.

4.2.3. Class 2 Transforms.

Transformations in Class 2, denoted by T_2, effect compression by encoding pixel values in *range*(*choice*(*domain*(T))) such that less storage space is required. However, the number of image pixels remains unchanged. Thus, $CR_d = 1$.

4.2.3.1. Assumption. Assume that an operation in a computational system incurs work $W_O(n)$ that is directly proportional to n.

The preceding assumption is reasonable for pointwise and global reduce operations, as well as for image-template operations that utilize templates whose size does not depend upon the image dimensions.

4.2.3.2. Lemma. Under constraint of Assumption 4.2.3.1, the computational efficiency η incurred by processing over the range space of T_2: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{X}}$ with operation $\bigcirc'$: $(\mathbf{G}^{\mathbf{X}})^{m} \rightarrow \mathbf{G}^{\mathbf{X}}$, as opposed to processing over *domain*(T_2) with operation $\bigcirc$: $(\mathbf{F}^{\mathbf{X}})^{m} \rightarrow \mathbf{F}^{\mathbf{X}}$, is given by

$$\eta = \frac{W_{\bigcirc}(siz(\mathbf{F}))}{W_{\bigcirc'}(siz(\mathbf{G}))} \propto \frac{siz(\mathbf{F})}{siz(\mathbf{G})} . \tag{146}$$

Proof. We previously showed that $CR(T_2) = siz(\mathbf{F})/siz(\mathbf{G})$. By the definitions of η and r, we have

$$\eta = \frac{W_{\bigcirc}(|\mathbf{X}|)}{r(\bigcirc, \bigcirc', |\mathbf{X}|) \cdot W_{\bigcirc}(|\mathbf{X}|)} = \frac{W_{\bigcirc}(siz(\mathbf{F}))}{W_{\bigcirc'}(siz(\mathbf{G}))} . \tag{147}$$

From Assumption 4.2.3.1, we have $W_{\bigcirc}(n) \propto n$. Thus, $\eta \propto siz(\mathbf{F})/siz(\mathbf{G})$, and the lemma holds. □

As a result, computation over *range*(T_2) derives its efficiency from the choice of the encoding alphabet $\mathbf{G}$, as well as from the work required by $\bigcirc'$ versus $\bigcirc$. This implies the possibility of achieving a computational efficiency $\eta > r$ that would be obtained via shortening the word length. The following example is illustrative.

4.2.3.3. Example. Assume that T_2: $\mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{X}}$ is a word truncation transform, expressed at medium granularity as ${}^{M}T_2$: $(\mathbf{Z}_8)^{\mathbf{X}} \rightarrow (\mathbf{Z}_4)^{\mathbf{X}}$. Let $\mathbf{a}, \mathbf{b} \in (\mathbf{Z}_8)^{\mathbf{X}}$ and $\mathbf{a}_c, \mathbf{b}_c \in (\mathbf{Z}_4)^{\mathbf{X}}$, such that $\mathbf{a}_c = T_2(\mathbf{a}) = \lfloor \mathbf{a}/2 \rfloor$. Let $\bigcirc = +_8$ denote addition modulo eight. Since $T_2(\bigcirc(\mathbf{a},\mathbf{b})) = \bigcirc'(\mathbf{a}_c,\mathbf{b}_c)$, we have that $\bigcirc'(\mathbf{a}_c,\mathbf{b}_c) = \lfloor \frac{1}{2}(\mathbf{a} +_8 \mathbf{b}) \rfloor$. Thus, $\bigcirc' = +_4$, addition modulo four. From Equation 147, if W is linear in the number of input bits, with the same proportionality constant for $\bigcirc$ and $\bigcirc'$ (for purposes of convenience) then $\eta = 1.5$, since $\eta \propto siz(\mathbf{F})/siz(\mathbf{G})$, where $siz(\mathbf{F}) = \log(8) = 3$ bits and $siz(\mathbf{G}) = \log(4) = 2$ bits.

4.2.4. Class 3 Transforms.

The complexity associated with transformations in Class 3 depends primarily upon the domain size of the input and output images. Per the definition of Class 3 given in Section 3.4, we assume that the wordlength of image pixel values, denoted by the *siz* function, remains unchanged.

4.2.4.1. Lemma. Under constraint of Assumption 4.2.3.1, the computational efficiency η incurred by processing over the range space of the Class 3 transform $T_3 : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ with operation $\bigcirc' : (\mathbf{F}^{\mathbf{Y}})^{\mathbf{m}} \rightarrow \mathbf{F}^{\mathbf{Y}}$, as opposed to processing over *domain*(T_3) with operation $\bigcirc : (\mathbf{F}^{\mathbf{X}})^{\mathbf{m}} \rightarrow \mathbf{F}^{\mathbf{X}}$, is given by

$$\eta = \frac{W_{\bigcirc}(|\mathbf{Y}| \cdot CR)}{\cdot\, W_{\bigcirc'}(|\mathbf{Y}|)} \propto CR. \tag{148}$$

Proof. The proof follows trivially from the definition of η and the fact that $CR(T_3) = |\mathbf{X}|/|\mathbf{Y}|$. □

4.2.4.2. Remark. Since the value set of *domain*(T_3) is preserved in *range*(T_3), it may be feasible to set the analogue $\bigcirc' = \bigcirc$. In such cases, the performance ratio $r_{(\bigcirc,\bigcirc')}$ is unitary, and $\eta = CR$. It is easily verified that if $W_{\bigcirc}$ is linear, then $\eta > CR$ is possible. Additionally, the preceding statement holds if $W_{\bigcirc}$ and $W_{\bigcirc'}$ are linear and sublinear in their input size, respectively.

The preceding complexity analyses of Classes 1–3 intuitively suggest that if both the value set and the point set of a source image were compressed, significant speedup could be realized. In practice, the speedup could exceed that achieved by point- or value-set compression alone. Furthermore, under constraint of the preceding remark, $\eta > CR$ could be possible. Such is the case with certain Class 4 transforms, as we show in the next section.

4.2.5. Class 4 Transforms.

We begin by assuming a decomposition of image operations and analogues over *range*(T_4) in terms of a greyscale function g and a spatial transform f. Then, we show how knowledge about the complexity of f and g can be combined to yield a generalized complexity function for Class 4 transforms.

4.2.5.1. Assumption Recall (from Section 3.3) that the Class 4 transforms $T_4 : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$. Assume the special case where a unary image operation $\bigcirc : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is comprised of a greyscale function $g : \mathbf{F} \rightarrow \mathbf{F}$ and a spatial transform $f : \mathbf{X} \rightarrow \mathbf{X}$, such that $\bigcirc = g \circ f$. That is, if $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, then $\bigcirc(\mathbf{a}) = g(\mathbf{a}) \circ f$.

Although the preceding decomposition provides a convenient vehicle for explaining methods of optimizing computational cost, but the decomposition itself may not exhibit optimal complexity. Thus, further optimization of the functions f and g may be required to achieve optimal performance.

4.2.5.2. Observation. Under constraint of Assumption 4.2.3.1, since composition is preserved under homomorphism, if T_4 is a homomorphism and $\bigcirc = g \circ f$, then it is possible that $\bigcirc'$, the analogue of $\bigcirc$ over *range*(T_4), could be decomposed as $\bigcirc' = g' \circ f'$. Here, $g' : \mathbf{G} \rightarrow \mathbf{G}'$ and $f' : \mathbf{Y} \rightarrow \mathbf{Y}$ denote the analogues of g and f over *range*(T_4), assuming that such analogues exist.

Given the preceding decomposition, we state the complexity of operations over *range*(T_4) in the following theorem.

4.2.5.3. Theorem. Under constraint of Assumption 4.2.3.1, the computational speedup of a Class 4 transform $T_4 : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ is given by

$$\eta = \frac{\mathrm{W}_g(siz(\mathbf{F}))}{\mathrm{W}_{g'}(siz(\mathbf{G}))} \cdot \frac{\mathrm{W}_f(|\mathbf{X}|)}{\mathrm{W}_{f'}(|\mathbf{Y}|)} \propto \mathrm{CR}\,. \tag{149}$$

Proof. The proof follows directly from Lemma 4.2.4.1 and the given assumptions, since

$$\eta = \frac{W_g(siz(\mathbf{F}))}{W_{g'}(siz(\mathbf{G}))} \cdot \frac{W_f(|\mathbf{X}|)}{W_{f'}(|\mathbf{Y}|)} \propto \frac{siz(\mathbf{F})}{siz(\mathbf{G})} \cdot \frac{|\mathbf{X}|}{|\mathbf{Y}|} = \mathrm{CR}. \tag{150}$$

Thus, the Lemma holds. □

Under the assumption that $\bigcirc$ and $\bigcirc'$ can be decomposed as stated, Equation 149 describes the general case of image transforms in Classes 1–4. However, the assumed decompositions may not exist, or may be intractable. Therefore, we provide proof when we employ the preceding theorem. A related comment pertaining to data security follows.

4.2.5.4. Remark. Assume that $q = CR_d$ and $p = CR_r$ are specific to a secure transform T, and would be concealed in practice. Further assume that the term *data security* means disclosing in *range*(T) as few of the properties of *domain*(T) as possible under theoretical or implementational constraints. For purposes of data security, we can write the numerator of Equation 149 as $T_g(p \cdot siz(\mathbf{G})) \cdot T_f(q \cdot |\mathbf{Y}|)$. Since Class 4 can be thought of as generalizing Classes 1–3, calculation of the computational speedup for any transform T in our taxonomy does not require knowledge of the properties of *domain*(T), such as $|\mathbf{F}|$ or $|\mathbf{X}|$. As discussed in Section 4.4, the disclosure of such properties may be crucial to data security.

We next present a brief discussion of salient issues pertaining to the implementational feasibility of computation over compressed data.

4.3. Feasibility of Compressive Computation.

We begin our analysis of compressive processing in an implementational scenario by considering a simple commutativity diagram that describes an isomorphic computational system with one transform and one unary operation (Section 4.5.1). In Section 4.5.2, we progress to an analysis of multiple image formats, and Section 4.5.3 expands our computational scenario to multiple operations and multiple images. Bounds upon the performance of such systems are given, together with a discussion of how such systems could be useful in practice.

4.3.1. Single Encoding Transform.

Assume that the primary goal of compressive processing is computational speedup. Now, reference the commutativity diagram of Figure 1, which schematically depicts an isomorphic computational system $X = (\{S, \bigcirc\}, \{T, \Theta\}, \{S', \bigcirc'\}, T^{-1})$ with unary operations $\bigcirc$ and $\bigcirc'$.

4.3.1.1. Assumption. Let the transform $T : \mathbf{F^X} \rightarrow \mathbf{G^Y}$ have an inverse $T^{-1} : \mathbf{G^Y} \rightarrow \mathbf{F^X}$. Let the unary image operation $\bigcirc : \mathbf{F^X} \rightarrow \mathbf{F^X}$ have an analogue $\bigcirc' : \mathbf{G^Y} \rightarrow \mathbf{G^Y}$, and let X as well as the images $\mathbf{a}$, $\mathbf{a}_c$, $\hat{\mathbf{a}}$, and $\hat{\mathbf{a}}_c$ be as defined previously.

Restricting the arity of $\bigcirc$ to m=1 for purposes of simplicity, we further assume that $T^{-1}(\bigcirc'(T(\mathbf{a}))) = \bigcirc(\mathbf{a})$. Thus, we have the following lemma that describes the complexity inherent in the system portrayed in Figure 1.

4.3.1.2. Lemma. Under constraint of Assumption 4.3.1.1, given work $W_{\bigcirc}(|\mathbf{X}|)$, the operation $T^{-1}(\bigcirc'(T(\mathbf{a})))$ exhibits an efficiency relative to the equivalent operation $\bigcirc(\mathbf{a})$ if and only if the following condition is satisfied:

$$W_{\bigcirc}(|\mathbf{X}|) < W_T(|\mathbf{X}|) + W_{\bigcirc'}(|\mathbf{Y}|) + W_{T^{-1}}(|\mathbf{Y}|). \tag{151}$$

Proof. The proof is obvious if one relabels the edges of the graph shown in Figure 1 with the corresponding complexity functions. □

We next discuss the more involved case of multiple encoding formats.

4.3.2. Processing with Multiple Encoding Transforms.

Implementationally, one may have a computational system where the group $(S, \bigcirc)$ is replaced by a heterogeneous algebra. In such an algebra, we assume that the various operand sets would each be expressed in terms of different compressive formats.

4.3.2.1. Assumptions. Let there exist ζ transforms denoted by $T_i: \mathbf{F}^{\mathbf{X}} \rightarrow (\mathbf{G}_i)^{\mathbf{Y}_i}$, $i \in \mathbf{Z}_\zeta$, where the sets $\mathbf{G}_i$ or $\mathbf{Y}_i$ may or may not be disjoint. Let each T_i have an inverse $T_i^{-1}: (\mathbf{G}_i)^{\mathbf{Y}_i} \rightarrow \mathbf{F}^{\mathbf{X}}$, $i \in \mathbf{Z}_\zeta$. For a given image operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ let a known analogue $\bigcirc_i': \mathbf{G}_i^{\mathbf{Y}_i} \rightarrow \mathbf{G}_i^{\mathbf{Y}_{\theta(i)}}$ over $range(T_i)$, where $\theta: \mathbf{Z}_\zeta \rightarrow \mathbf{Z}_\zeta$ is an indexing function. If we restrict the arity of each operation to m=1, for purposes of simplicity, such a system can be described by Figure 4.

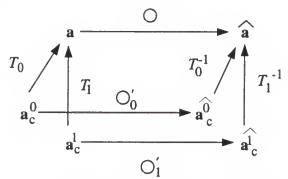


Figure 4. Commutativity diagram of a compressive computational system with multiple encoding transforms T_i and multiple analogues $\bigcirc_i'$ of image operation $\bigcirc$.

4.3.2.2. Lemma. Under constraint of Assumption 4.3.2.1, given the time complexity $T_{\bigcirc}(|\mathbf{X}|)$, the operation $T_i^{-1}(\bigcirc_i'(T_i(\mathbf{a})))$, $i \in \mathbf{Z}_\zeta$, exhibits a speedup relative to $\bigcirc(\mathbf{a})$ if and only if the following condition is satisfied:

$$W_{\bigcirc}(|\mathbf{X}|) < W_{T_i}(|\mathbf{X}|) + W_{\bigcirc_i'}(|\mathbf{Y}_i|) + W_{T_i^{-1}}(|\mathbf{Y}|), \; i \in \mathbf{Z}_\zeta . \tag{152}$$

Proof. The proof is obvious if one relabels the operations in Figure 4 with the corresponding complexity functions, then considers the commutativity diagram as a weighted graph. By the definition of computational speedup η, the i-th path from $\mathbf{a}$ to $\hat{\mathbf{a}}$, via the edge labelled $\bigcirc_i'$, must satisfy Equation 152 for a speedup to occur. □

4.3.3. Processing with Multiple Transforms and Multiple Operations.

The structure depicted in the graph of Figure 4 can be extended to include multiple image operations $\bigcirc$, as well as the multiple compression formats depicted schematically in Figure 4. We formalize this concept as follows.

4.3.3.1. Assumption. Let an image processing algorithm be comprised of ψ image operations $\bigcirc_j : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$, $j \in \mathbf{Z}_\psi$. Here, unary operations are employed for purposes of brevity but without loss of generality. Let there exist ζ transforms $T_i : \mathbf{F}^{\mathbf{X}} \rightarrow (G_i)^{\mathbf{Y}_i}$, $i \in \mathbf{Z}_\zeta$, where the sets G_i may or may not be disjoint, and likewise for $\mathbf{Y}_i$. Let each T_i have an inverse $T_i^{-1} : (G_i)^{\mathbf{Y}_i} \rightarrow \mathbf{F}^{\mathbf{X}}$, $i \in \mathbf{Z}_\zeta$. Further assume that, for a each image operation $\bigcirc_j$ there exist ζ known analogues of form $\bigcirc'_{j,i} : G_i^{\mathbf{Y}_i} \rightarrow G_i^{\mathbf{Y}_{\theta(i)}}$ that compute over *range*(T_i), where $\theta : \mathbf{Z}_\zeta \rightarrow \mathbf{Z}_\zeta$ denotes an indexing function. Additionally, denote a transform that converts from the l-th to the m-th format by T_{lm}, $l, m \in \mathbf{Z}_\zeta$. Letting $\psi = 2$ and $\zeta = 2$ for purposes of simplicity, we obtain the computational system diagrammed schematically in Figure 5.

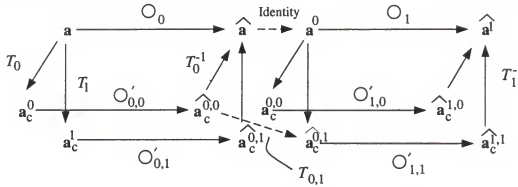


Figure 5. Commutativity diagram of a compressive computational system with multiple encoding transforms T_i, $i \in \mathbf{Z}_\zeta$, and multiple analogues $\bigcirc'_{j,i}$, $j \in \mathbf{Z}_\psi$ of the j-th image operation $\bigcirc_j$, where $\zeta = 2$ and $\psi = 2$.

4.3.3.2. Notation. Assuming the existence of an indexing function $k : \mathbf{Z}_\psi \rightarrow \mathbf{Z}_\zeta$, we denote the work associated with each edge in the graph of Figure 5, as follows:

$W_{\bigcirc_j}(\lvert\mathbf{X}\rvert)$	the j-th image operation
$W_{T_{k(j)}}(\lvert\mathbf{X}\rvert)$	the j-th encoding transform that yields the k(j)-th encoding format
$W_{\bigcirc'_{j,k(j)}}(\lvert\mathbf{Y}_{j,k(j)}\rvert)$	an analogue to the j-th image operation over *range*$(T_{k(j)})$
$W_{T_{k(j),k(j+1)}}(\lvert\mathbf{X}\rvert)$	the j-th conversion $T_{l,m}\left(\mathbf{a}_c^j\right) = T_m\left(T_l^{-1}\left(\mathbf{a}_c^j\right)\right)$, $l, m \in \mathbf{Z}_\zeta$

Such formalisms, together with the graph of Figure 5, intuitively suggests the following theorem.

4.3.3.3. Theorem. Under constraint of Assumption 4.3.3.1, the k-th sequence of transform-regime analogues, denoted by

$$\bigcirc'_{0,k(0)}, \bigcirc'_{1,k(1)}, \cdots, \bigcirc'_{j,k(j)}, \cdots, \bigcirc'_{\psi-1,k(\psi-1)}, j \in \mathbf{Z}_\psi, \tag{153}$$

exhibits a computational efficiency, relative to the corresponding sequence of image operations $S_\psi = \{\bigcirc_0, \bigcirc_1, \ldots, \bigcirc_\psi\}$ if and only if there exists a mapping $k : \mathbf{Z}_\psi \to \mathbf{Z}_\zeta$ such that the computational work incurred in traversing a path from image $\mathbf{a}$ to image $\hat{\mathbf{a}}_\psi$ is less than the work incurred by computing the sequence S_ψ.

Proof. The proof follows trivially from the existence of the shortest path algorithm, from which one obtains the function k. □

4.3.3.4. Remark. It is well known that the cost of computing the shortest path over a weighted digraph of N edges is $\mathbf{O}(N^3)$. The graph depicted in Figure 5 has ψ operations, and ζ transforms, which yields the following edge count:

1. ψ operations yields ψ-1 edges on the spine of the graph;
2. For each of ψ operations and each of ζ transforms, there are three edges, namely, T, $\bigcirc'$, and T^{-1}, yielding $3\psi\zeta$ edges; and
3. For ψ-1 operations, there are ζ^2-ζ edges involved in the format conversion transforms, since conversion between like formats merely requires an identity.

Thus, the total number of edges in the graph is given by

$$\begin{aligned} N(\psi,\zeta) &= \psi + (\psi - 1)(\zeta^2 - \zeta) + 3 \cdot \psi \cdot \zeta \\ &= \psi \cdot (\zeta^2 - \zeta + 3\zeta + 1) - \zeta^2 + \zeta \\ &= (\psi + 1) \cdot \zeta^2 + (2\zeta - 1) \cdot \zeta + \psi \\ &= \mathbf{O}(\psi \cdot \zeta^2). \end{aligned} \tag{154}$$

As a result, application of the shortest-path algorithm to the graph of Figure 3 incurs work of $\mathbf{O}\left((\psi \cdot \zeta^2)^3\right) = \mathbf{O}(\psi^3 \cdot \zeta^6)$ operations. Although the number ζ of available compression

transforms is relatively small, the cost may be impractical for an algorithm which has a large number of steps ψ or a system that processes over many formats ζ.

Note also that the preceding theorem presupposes that $|\mathbf{Y}_{j,k(j)}|$ is known a priori for all $j \in \mathbf{Z}_\psi$. When such static domains are impractical, as is often the case when processing is restricted to areas of interest whose domains are data-dependent, we make the following provision for a greedy optimization procedure.

4.3.3.5. Theorem. Under constraint of Assumption 4.3.3.1, further assume that the compressed domains exhibit sizes denoted by $|\mathbf{Y}_{j,k(j)}|, j \in \mathbf{Z}_\psi$, which are not known a priori. Then, the work $W(\psi,\zeta)$ associated with a simple path along the graph of Figure 5, which path may exhibit minimum computational cost with respect to the sequence of image operations $S_\psi = \{\bigcirc_0, \bigcirc_1, \ldots, \bigcirc_\psi\}$, is computed via the following greedy algorithm. For purposes of simplicity, let each operation (e.g., T, $\bigcirc'$, T^{-1}) denote the computational work associated with that operation (e.g., W_T, $W_{\bigcirc'}$, $W_{T^{-1}}$).

Step 1. Compute the minimum work component of the first edge of Figure 5, corresponding to $\bigcirc_0$, as

$$W_0(\psi,\zeta) = \bigwedge\left(\bigcirc_0(\mathbf{a}), \bigwedge_{i\in\mathbf{Z}_\zeta}\left[T_i(|\mathbf{X}|) + \bigcirc'_{0,i}(|\mathbf{Y}_{0,i}|) + \right.\right. \\ \left.\left. + \bigwedge_{m\in\mathbf{Z}_\zeta}\left(T_m^{-1}(|\mathbf{X}|), \bigwedge_{l\in\mathbf{Z}_\zeta} T_{i,l}(|\mathbf{Y}_{1,i}|)\right)\right]\right). \tag{155}$$

Step 2. Compute the minimum work component of each edge of Figure 5 that corresponds to $\bigcirc_1$ through $\bigcirc_{\psi-2}$ as

$$W_j(\psi,\zeta) = \bigwedge\left(\bigcirc_j(\hat{\mathbf{a}}^j), \bigwedge_{i\in\mathbf{Z}_\zeta}\left[\bigcirc'_{j,i}(|\mathbf{Y}_{j,i}|) + \right.\right. \\ \left.\left. + \bigwedge_{m\in\mathbf{Z}_\zeta}\left(T_m^{-1}(|\mathbf{Y}_{j,m}|), \bigwedge_{l\in\mathbf{Z}_\zeta} T_{i,l}(|\mathbf{Y}_{j,i}|)\right)\right]\right). \tag{156}$$

Step 3. Compute the minimum work incurred by the edge in Figure 5 that corresponds to $\bigcirc_{\psi-1}$, as

$$W_{\psi-1}(\psi,\zeta) = \bigwedge\left(O_0\left(\hat{\mathbf{a}}^{\psi-1}\right), \bigwedge_{i\in\mathbf{Z}_\zeta}\left[O'_{j,i}(|\mathbf{Y}_{j,i}|) + \right.\right. \\ \left.\left. + \bigwedge_{m\in\mathbf{Z}_\zeta}\left(T_m^{-1}(|\mathbf{Y}_{j,m}|), \bigwedge_{l\in\mathbf{Z}_\zeta} T_{i,l}(|\mathbf{Y}_{j,i}|)\right)\right]\right). \tag{157}$$

Step 4. Sum the preceding work to obtain the total work, which is given by

$$W(\psi,\zeta) = W_0(\psi,\zeta) + \left(\sum_{j=1}^{\psi-2} W_j(\psi,\zeta)\right) + W_{\psi-1}(\psi,\zeta). \tag{158}$$

Proof. The proof follows directly from the construction of Figure 5 and the development of Sections 4.3.1–4.3.3. □

4.3.3.6. Remark. Assume that we have appropriate indexing functions, with which we compute the indices of the operations T, $\bigcirc'$, and T^{-1} in Steps 1–3 of the preceding algorithm. For purposes of argument, assume that a computation that incurs minimum work would correspond to the shortest path on the graph of Figure 5. One can thus construct a sequence of operations that may exhibit lower computational cost than the sequence of image operations S_ψ.

In practice, we must qualify the foregoing assertion by stating the well-known observation that greedy optimization will not necessarily yield a minimum-work sequence of compressive (or noncompressive) operations which computes $\hat{\mathbf{a}}$ (or an approximation thereof) from $\mathbf{a}$. Additionally, the overhead incurred by the greedy optimization procedure is computed similar to $W(\psi,\zeta)$, and is of order $\psi\cdot\zeta^2$. This estimate can be verified by an inspection of Theorem 4.3.3.5. As expected, the greedy algorithm exhibits lower cost than the shortest-path approach, by a factor of $\mathbf{O}(\psi^2\cdot\zeta^4)$.

Various greedy optimization procedures have been developed that exhibit a higher probability of an optimal solution as the lookahead increases [65]. However, such greedy algorithms have as a common deficit the inability to *consistently* reach a global optimum.

As a result, the greedy optimization outlined in Theorem 4.3.3.5, while exhibiting lower overhead than the shortest-path algorithm, will not necessarily yield the optimal computational path through the graph depicted in Figure 5. The subject of optimal computation over such networks will be investigated in detail in future research.

We next consider the issue of data security, which is another potential advantage of compressive processing, and is key to the design of computational systems that process encrypted data.

4.4. Data Security.

This section overviews a review of data security theory, which we express in terms of properties of mappings. We show how each transform class in the taxonomy of Section 3.3 conceals or discloses properties of the transform domain. We then apply our data security theory, together with theory developed in Chapters 3 and in the preceding part of Chapter 4, to describe computation over encrypted data.

Throughout this chapter, we assume the following implementational scenario. First, given an encryptive transform T, we define the *data security of* T at a high level in terms of information about a plaintext $\mathbf{a}$ that is obtainable from the corresponding ciphertext $\mathbf{a}_c = T(\mathbf{a})$. If no properties of $\mathbf{a}$ can be inferred from $\mathbf{a}_c$ by a given cryptanalytic method, then we say that T is secure with respect to that method. Additionally, if one cannot infer *operationally salient* properties of a from $\mathbf{a}_c$, then we say that T is operationally secure. A key problem, however, is determining which properties are salient with respect to a given operational scenario.

Second, we assume that an image processing system is used by *non-privileged users* and is maintained by *privileged users* (e.g., programmers and systems administrators). The non-privileged users can only specify image processing algorithms in the image domain. For example, one would construct an algorithm from operations such as the image sum,

Sobel edge operator, etc., given a library of permissible operations. If so required, non-privileged users can be prevented from viewing imagery in either encrypted or decrypted form. Thus, non-privileged users think they are working with image operations, when they actually may be computing over encrypted data (i.e., with operations that are transform-regime analogues of the image operations). Additionally, non-privileged users can be prevented from becoming security threats by not allowing them to see the imagery they are processing. For the purposes of this development, such constraints are assumed to exist where appropriate. For purposes of simplicity, we further assume that such constraints cannot be altered by any class of users. Therefore, we assume that the non-privileged user's only salient capability is the specification and invocation of algorithms constructed from a list of permissible operations.

In contrast, we assume that the privileged users are given a library of plaintext operations, as well as a library containing the corresponding analogous operations over ciphertext. Using such operations the privileged users specify and construct the building blocks to which the non-privileged users are allowed access. Now, the privileged users may not know the internal workings of either the encryption transform(s) or the ciphertext operations, but may use such mappings arbitrarily. Thus, the system is vulnerable if the privileged users are adversaries and can employ one or more encryptive analogues $\bigcirc'$ to exercise the output of a given encryption T and thus attack T. This technique will be discussed subsequently.

Third, we assume that the privileged users may apply T and $\bigcirc'$ to various input images, which would enable them to guess T or $\bigcirc'$ by analysis of the resultant output. In particular, we assume that if privileged users are adversaries, then T may be guessed at via cryptanalytic techniques.

Fourth, we assume that the implementations of all transforms T and ciphertext operations $\bigcirc'$ are maintained at levels of medium and fine granularity (i.e., data structures

and functionality, respectively) by trustworthy personnel. Thus, only privileged users are admitted to the role of adversary.

Our implementational goal is encryptive processing with the highest level of security possible under the foregoing constraints. That is, no user should be able to determine the encryption T by using one or more analogous operations $\bigcirc'$, or by analysis of transform inputs and outputs. In this somewhat artificial scenario, we do not consider specific implementational problems such as key distribution or tapping of data distribution lines. Such implementational issues are outside the core issue of the security of T and $\bigcirc'$. Instead, since this study is primarily theoretical, we prefer to concentrate upon the security of mappings inherent in T and $\bigcirc'$. We thus direct the reader to Reference 60 for an excellent implementational discussion of computer system security.

In this study, the key questions pertaining to data security are as follows:

1. How secure is a given encryption transform T, i.e., how easily can T be guessed, and under what conditions?
2. How secure are the operations $\bigcirc'$ that process over *range*(T), i.e., what properties of $\mathbf{a}$ are disclosed in $\bigcirc'(T(\mathbf{a}))$?

In order to answer such questions, we develop several concepts.

First, we assume that an encryption transform or encryption conceals properties of the plaintext input in the ciphertext output. For the purposes of this study, an encryption is deemed *insecure* if salient properties of the transform's input (plaintext) are disclosed in the transform's output (ciphertext). We generalize the preceding convention theoretically in terms of a transform's domain and range spaces. Accordingly, in this section we develop theory that describes the properties of a mapping, beginning with cardinality. Then, we describe properties of sets and mappings that pertain to analyses presented in Chapters 6–9. We close with a brief discussion of the propagation of transform domain properties into the transform's range space.

4.4.1. Cardinality of Mappings.

The success of a cryptanalytic attack depends upon a knowledge of the work involved in attacking a discrete mapping. In the case of finite discrete images, a knowledge of the image domain and range spaces is key to furnishing useful clues to the plaintext contents or the keyspace. At each step of the cryptanalytic process, we thus try to guess salient properties of the mappings that may be involved in the encryption. The first step in this guessing process is the determination of the domain cardinality (number of pixels or symbols) and range cardinality (length of the alphabet), where possible. We present relevant theory, as follows.

4.4.1.1. Definition. The Cartesian product $S^m = \prod_{i=1}^{m} S$ exhibits cardinality $|S^m| = |S|^m$.

We state the following definition, in order to make plain to the reader when cardinality is defined.

4.4.1.2. Definition. If M is a mapping, then one of the following conditions holds:

(i) If *domain*(M) and *range*(M) are finite, then M and $|M|$ are finite.

(ii) If *domain*(M) or *range*(M) are countably infinite, then M and $|M|$ are infinite but countable.

(iii) If *domain*(M) or *range*(M) are uncountable, then M and $|M|$ are undefined.

4.4.1.3. Definition. The cardinality of a finite mapping M is given by $|M| = |range(M)|^{|domain(M)|}$.

4.4.1.5. Example. If a finite mapping $M : \mathbf{Z}_m \rightarrow (\mathbf{Z}_2)^{\lceil \log_2 m \rceil}$, then

$$\begin{aligned} |M| &= |range(M)|^{|domain(M)|} \\ &= \left|(\mathbf{Z}_2)^{\lceil \log_2 m \rceil}\right|^{|\mathbf{Z}_m|} = \left(|\mathbf{Z}_2|^{\lceil \log_2 m \rceil}\right)^{|\mathbf{Z}_m|} \\ &= \left(2^{\lceil \log_2 m \rceil}\right)^m \approx m^m . \end{aligned} \tag{159}$$

It follows that $(m+1)^m \leq |M| \leq m^m$.

4.4.2. Circulant Sets and Mappings.

Circulant sets and mappings are important in cryptography, for example, in modulo arithmetic. We therefore present the following definitions.

4.4.2.1. Definition. Let S be an indexed set, where $f: S \rightarrow \Lambda$ is an indexing function, and the index set Λ is totally ordered, e.g., $\Lambda \subset \mathbf{N}$. We customarily say that an element $g \in S$ has

(i) an *immediate predecessor* $f \in S$ iff $f(f)$ precedes $f(g)$ in the order inherent in Λ, and

(ii) an *immediate successor* $h \in S$ iff $f(g)$ precedes $f(h)$ in the order inherent in Λ.

4.4.2.2. Definition. A *circulant set* is an indexed set S whose index set Λ satisfies the following conditions:

(i) If $\Lambda = \mathbf{Z}_{|S|}$ with $n = |\Lambda|$, then $f(S_{n-1})$ is the immediate predecessor of $f(S_0)$ in the ordered set S, and

(ii) If $\Lambda = \mathbf{Z}_{|S|}$ with $n = |\Lambda|$, then $f(S_0)$ is the immediate successor of $f(S_{n-1})$ in the ordered set S.

Thus, in a circulant set, every element is a predecessor of every other element. This concept has important implications for modular arithmetic, as shown in Chapters 6–9. We next discuss two important properties of circulant mappings.

4.4.2.3. Definition. A *circulant mapping* has range and domain spaces that are circulant sets.

4.4.2.4. Definition. A *circulant function* is a circulant mapping of arity m-to-1.

In Chapter 3, we discussed the preservation of certain properties of sets. In the following definition and example, we elaborate the concept of properties in order to facilitate the subsequent definition of oracles.

4.4.2.5. Definition. For purposes of data security, a *property* of a mapping is defined by the statement $\alpha\ \beta$, where α denotes an action or state of being, and β denotes a property of a set that constitutes the mapping.

4.4.2.6. Example. If a transform T is a homomorphism, then we express the property of homomorphism as α = *is* and β = *homomorphism.*

Additionally, given the transform class T_1, we say that T_1 *preserves* the *size* of an input image, such that $|domain(choice(range(T_1)))| = |domain(choice(domain(T_1)))|$. Here, α = *preserves* and $\beta = size(domain(T_1)) = |domain(choice(domain(T_1)))|$.

4.4.3. Graphs and Matrix Representations.

Properties of a given mapping M can be described in terms of the graph of M, together with the adjacency matrix of the graph. For example, such representations are especially useful for cryptanalysis of a mapping M that employs synthetic arithmetic, where an adversary would attempt to discover properties such as *is associative* or *is commutative* by scrutinizing some representation of the inputs and outputs of M.

4.4.3.1. Definition. The *graph G of a mapping M* is given by

$$G(M) = \{(x,y): x \in domain(M), y = M(x)\}. \tag{160}$$

For those readers who are unfamiliar with the concept of graphs, we note that the graph $G(M) = (V,E)$ such that the *vertex set* V of G is given, per the preceding definition, by $V = domain(M) \cup S$, where $S \subseteq range(M)$ and the *edge set* $E \subseteq V \times V$.

4.4.3.2. Definition. Let $G = (V,E)$ denote the graph of mapping M. Let the isomorphism $f: V \rightarrow \Lambda$ index the elements of G such that $G' \sim (V', E') = (\Lambda, \Lambda \times \Lambda)$, where $(\sim)$ signifies isomorphism and Λ denotes an arbitrary indexing set. Then $\mathbf{a} \in \{0,1\}^{\Lambda \times \Lambda}$, the *reachability image of G*, is defined as

$$\mathbf{a}(\lambda_1,\lambda_2) = \begin{cases} 1 & \text{if } (\lambda_1,\lambda_2) \in E' \\ 0 & \text{otherwise} \end{cases} . \tag{161}$$

We can similarly represent the mapping M as a weighted digraph $G(M)$, from which we derive a matrix representation of M, as follows.

4.4.3.3. Definition. Let $G = (V,E)$ be the graph of g, a binary function over a subset of V, and let the indexing function $f: V \rightarrow \Lambda$. Denote $\hat{f}: V \times V \rightarrow \Lambda \times \Lambda$, such that $\hat{f}(e) = (f(p_1(e)), f(p_2(e)))$, $e \in E \subseteq V \times V$. Then, the *matrix representation of g*, denoted by $\mathbf{g} \in \Lambda^{\Lambda \times \Lambda}$, is given by

$$\mathbf{g} \equiv \{(\mathbf{x}, g(\mathbf{x})): \mathbf{x} \in E, \mathbf{h}(\mathbf{x}) \in \mathit{range}(M) \subset V\}. \tag{162}$$

Such matrix representations (MRs) of a function are useful constructs for testing properties such as commutativity and associativity. Given g, the discrete computation of the MR of g can facilitate the determination of numerous properties and effects that are useful in cryptanalysis. As previously mentioned, this technique is especially applicable to the analysis of operations over synthetic number systems. We give the following lemmas as examples.

4.4.3.4. Lemma. The matrix representation of a commutative function over S is symmetric about its major diagonal.

Proof. Assume, for purposes of convenience, that the operation $\bigcirc : S \times S \to S$ is commutative, and that the indexing function $f: S \to \mathbf{N}$. Denote $i = f(f)$, $j = f(g)$, and $k = f(h)$, and let the matrix representation of $\bigcirc$ be given by $\mathbf{a}(i,j) = f(f \bigcirc g)$, $\forall f, g \in S$. We then write

$$f \bigcirc g = g \bigcirc f \text{ implies that}$$

$$f(f \bigcirc g) = f(g \bigcirc f) \tag{163}$$

$$= \mathbf{a}(i,j) = \mathbf{a}(j,i), \forall f, g \in S.$$

Thus, the matrix representation $\mathbf{a}$ is symmetric about its major diagonal. □

4.4.3.5. Lemma. Let the operation $\bigcirc : S \times S \to S$ be associative, and let the indexing function $f: S \to \Lambda \subset \mathbf{N}$. The matrix representation of $\bigcirc$ is given by

$$\mathbf{a}(\mathbf{a}(i,j),k) = \mathbf{a}(i,\mathbf{a}(i,j)) \ \forall i,j,k \in f(S). \tag{164}$$

Proof. Assume that (a) the operation $\bigcirc : S \times S \to S$ is associative, (b) the indexing function $f: S \to \Lambda \subset \mathbf{N}$ is a bijection, and (c) that $\mathbf{a} \in \Lambda^{\Lambda \times \Lambda}$. Further assume that $\mathbf{a}(\mathbf{a}(i,j),k) = \mathbf{a}(i,\mathbf{a}(i,j))$ $\forall i,j,k \in f(S)$. Denote $i = f(f)$, $j = f(g)$, and $k = f(h)$. By applying $f^{-1}(\mathbf{a}(i,j)) = \bigcirc(f^{-1}(i), f^{-1}(j))$, to Equation 164 we obtain

$$f^{-1}(\mathbf{a}(\mathbf{a}(i,j),k)) = f^{-1}(\mathbf{a}(i,\mathbf{a}(j,k))) \text{ implies that}$$

$$\begin{aligned}
(f \bigcirc g) \bigcirc h &= \bigcirc(f^{-1}(\mathbf{a}(i,j)), f^{-1}(k)) \\
&= \bigcirc(f^{-1}(i), f^{-1}(\mathbf{a}(j,k))) \\
&= \bigcirc(\bigcirc(f^{-1}(i), f^{-1}(j))f^{-1}(i), h) \\
&= \bigcirc(f, \bigcirc(f^{-1}(j), f^{-1}(k))) \\
&= \bigcirc(f, \bigcirc(g,h)) \\
&= f \bigcirc (g \bigcirc h).
\end{aligned} \tag{165}$$

Thus, the function $\bigcirc$ is associative and the lemma holds. □

4.4.3.6. Lemma. Let the operations $\bigcirc, \gamma : S \times S \to S$, where γ distributes over $\bigcirc$, and let the indexing function $f : S \to \Lambda \subset \mathbf{N}$. The matrix representation of $\bigcirc$ and γ, denoted by $\mathbf{a} \in \Lambda^{\Lambda \times \Lambda}$ and $\mathbf{b} \in \Lambda^{\Lambda \times \Lambda}$, respectively, is given by

$$\mathbf{a}(i,\mathbf{b}(j,k)) = \mathbf{b}(\mathbf{a}(i,j),\mathbf{a}(i,k)) \ \forall i,j,k \in f(S). \tag{166}$$

Proof. Assuming that the givens hold, the proof consists of transforming both sides of the distributivity law $f \bigcirc (g \gamma h) = (f \bigcirc g) \gamma (f \bigcirc h)$, $\forall f,g,h \in S$, by applying f^{-1}, in a manner symmetric to that of Lemma 4.4.3.5. □

4.4.3.7. Lemma. Let the operations $\bigcirc, \gamma : S \times S \to S$ be heteroassociative, and let the indexing function $f : S \to \Lambda \subset \mathbf{N}$. The matrix representation of $\bigcirc$ and γ, denoted by $\mathbf{a} \in \Lambda^{\Lambda \times \Lambda}$ and $\mathbf{b} \in \Lambda^{\Lambda \times \Lambda}$, respectively, is given by

$$\mathbf{b}(\mathbf{a}(i,j),k) = \mathbf{a}(i,\mathbf{b}(j,k)) \ \forall i,j,k \in f(S). \tag{167}$$

Proof. Assuming that the givens hold, the proof consists of transforming both sides of the definition of hetereoassociativity $f \bigcirc (g \gamma h) = (f \bigcirc g) \gamma h$, $\forall f,g,h \in S$, by applying f^{-1} in a manner symmetric to that of Lemma 4.4.3.5. □

There are certain abstract properties of mappings that are useful in cryptanalysis, such as the histogram of an image, input size, etc., which are easily computed. However, another interesting property of two mappings is their similarity. For example, assume that one has a known encryption $T_a(\mathbf{a},\mathbf{k})$, where $\mathbf{k}$ denotes the key, that produces a result that is similar to the encryption $T_b(\mathbf{a},\mathbf{k})$ whose internal operation is unknown. It is possible to apply T_a and T_b to identical inputs, and compare the outputs via methods such as correlation. If this process can be pursued to exhaustion, then one has a complete comparison of the mappings inherent in T_a and T_b with respect to a similarity measure such as the correlation coefficient. Otherwise, T_a and T_b can be characterized as partial functions and can be correlated to obtain a *partial similarity measure*. Given such descriptions, one can construct the graphs of

T_a and T_b, which can be compared using graph similarity functions to provide information on the resemblance of the unknown mapping to the known. We provide theoretical support and an example, as follows.

4.4.3.8. Definition. Let M be a mapping, and let $\xi : domain(M) \times range(M) \rightarrow \mathbf{R}$ denote a *similarity function* that returns a measure of similarity between $domain(M)$ and $range(M)$. Let $t \in \mathbf{R}$ denote a *similarity threshold.* Either one of the following statements holds, or both (i) and (ii) hold:

(i) M is termed a *correlated mapping* (or is simply called *correlated*) if $\xi(S, M(S)) \geq t, \forall S \in domain(M)$,

(ii) $M(S)$ is said to *correlate with* S if $\xi(S, M(S)) \geq T$, for some $S \in domain(M)$, or

(iii) M is *uncorrelated.*

It is easily verified that the preceding definition can include comparisons of data structures such as graphs. However, for purposes of simplicity, we couch our expression of the similarity measure between mappings in terms of the familiar operation of correlation. In Parts II and III, we further discuss the concept of similarity between mappings, and next define a simple model of data security.

4.4.4. A Set-Theoretic Convention for Data Security

For purposes of discussion, assume that we have a "magic" computer program that can determine the properties of a mapping, given a set of operational constraints. We call such a program an *oracle.* The concept of oracles underlies cryptanalytic theory, but is rarely discussed openly, since specification of an oracle could reveal one's encryption capabilities to an adversary. In this section, we formulate a simple oracle for purposes of defining a model of data security that is based upon properties of a mapping. In this discussion, our oracles are not meant to be comprehensive implementationally, but are kept simple to illustrate technique.

We begin by asserting that an encryption T may disclose information about a plaintext $\mathbf{a}$ in the ciphertext $T(\mathbf{a})$. We then show how an oracle can be applied to a mapping (which implies the mapping's input and output) to deduce properties of T that would be useful to a cryptanalyst. We further extend our model to discuss the security of compressive or encryptive computational systems.

4.4.4.1. Definition. Let a simple *oracle* be denoted by an operation $\mathfrak{O}$: $\mathcal{A} \times \mathcal{T} \rightarrow (\mathbf{Q} \times \mathbf{R}^2)^n$ that is not omniscient, and consists of

(i) an algebra $\mathcal{A} = (\mathcal{F}, \mathcal{O})$ that contains operands in $\mathcal{F}$ and operations in $\mathcal{O}$,

(ii) a mapping T in $\mathcal{T}$, where *range*(T),*domain*$(T) \in \mathcal{F}$, and

(iii) a tuple of n elements, where each element consists of a property $q \in \mathbf{Q} = \mathcal{P}(T)$ together with two real numbers that specify a measure such as the probability of correctly determining q, as well as the computational error (or inaccuracy in the error estimate) associated with that determination (where applicable).

For the purposes of this section's discussion, we restrict the preceding definition to constrain the oracle to a scrutiny of properties of T that are known to exist in *domain*(T), and may or may not be evident in *range*(T).

4.4.4.2. Assumption. Given a cryptanalytic system that is characterized by an algebra A in $\mathcal{A}$, an encryption T in $\mathcal{T}$, and an oracle $\mathfrak{o} \in \mathfrak{O}$, we assume for purposes of discussion that $\mathfrak{o}(A, T)$ returns a list of properties q_i, i=1..n, of T obtained from the scrutiny of *range*(T), given knowledge about *domain*(T).

4.4.4.3. Example. Let an oracle $\mathfrak{o} \in \mathfrak{O}$ have in its domain the algebra $A = \left((\mathbf{Z}_m)^{\mathbf{X}}, \{\mathbf{h}, \mu\}\right)$, where the finite domain $\mathbf{X} \subset \mathbf{Z}^2$, the histogram $\mathbf{h}$: $(\mathbf{Z}_m)^{\mathbf{X}} \rightarrow \mathbf{N}^{\mathbf{Z}_m}$, and μ: $(\mathbf{Z}_m)^{\mathbf{X}} \rightarrow \mathbf{Z}_m$ denotes the image mean truncated to the nearest integer. Let $\mathfrak{o}$ accept a bijection T: $\mathbf{Z}_m \rightarrow \mathbf{Z}_m$ in $\mathcal{T}$, and return a tuple $((q_1, p_1, \epsilon_1), (q_2, p_2, \epsilon_2))$.

For example, $q_1 \in \mathbf{N}^{\mathbf{Z}_m}$ could denote *preserves input histogram.* Here, the probability $p_1 = 1$ and error $\epsilon_1 = 0$, since the histogram of an integer-valued image of finite size can be computed exactly. Since T is an isomorphism, if $\mathbf{a} \in (\mathbf{Z}_m)^{\mathbf{X}}$, then

$$\mathbf{h}(T(\mathbf{a}))(i) = \mathbf{h}(\mathbf{a})(T(i)), \; i \in \mathbf{Z}_m, \; \forall \mathbf{a} \in \mathit{domain}(T). \tag{168}$$

Continuing with property q_2, which we assume denotes *does not preserve mean*, the mean μ of an input image $\mathbf{a}$ would not necessarily be preserved, since $\mu(T(\mathbf{a})) \neq \mu(\mathbf{a}), \forall \mathbf{a} \in \mathit{domain}(T)$, for all certain substitutions T. For example, if $\mathit{range}(\mathbf{a}) = \{0,1\}$, and $\mathbf{a}$ contains unequal numbers of zeroes and ones, then the transform $T(\mathbf{a}) = |\mathbf{a} - 1|$ is a bijection but does not necessarily preserve the mean. Thus, the probability $p_2 = 1$ and error $\epsilon_1 = 0$, since it is certain that *does not preserve mean* is a property of T.

4.4.4.4. Remark. We have thus far discussed only a very restricted type of oracle, which could determine properties such as the disclosure of cardinality, order, moments, or the histogram of an image in $\mathit{domain}(T)$ via scrutiny of T's input and output. In practice, a simple oracle, such as that given in Definition 4.4.4.1, could be used to determine the preservation of such properties by an encryption transform. In contrast, a more capable oracle might accept clues about plaintext $\mathbf{a}$ (e.g., semantics, word order, letter n-gram frequency) and deduce further information about the corresponding properties of the ciphertext $T(\mathbf{a})$ (e.g., how T affects plaintext semantics and word order).

In keeping with our previous assumptions, we shall employ the simple oracle to define the concepts of *disclosure* and *concealment* of properties of $\mathit{domain}(T)$ in $\mathit{range}(T)$. More complex oracles will be discussed in Parts II and III.

4.4.4.5. Definition. Let an oracle $o \in \mathfrak{O}$ employ an algebra $A \in \mathcal{A}$ that contains operators and operands employed in cryptanalysis, and let o accept an encryption $T \in \mathcal{T}$. The properties of T that are *disclosed* or *leaked* to o's scrutiny of T are expressed as $\mathcal{D}(T) = p_1(o(A, T))$. Here, the existence of o is implied in the notation $\mathcal{D}(T)$, unless otherwise stated.

For example, T's security could be enhanced by minimizing $|\mathcal{D}(T)|$.

4.4.4.6. Definition. Properties of T that are *hidden* or *concealed* from o's scrutiny of T are given by $\mathcal{H}(T) = \mathcal{P}(T) \setminus \mathcal{D}(T)$.

4.4.4.7. Remark. The preceding definitions are convenient for two reasons. First, we properly limit the capabilities of oracles to non-omniscience. Since an omniscient oracle would be able to solve all problems, but Göedel has shown that not all functions are computable, there is a contradiction when omniscience is attributed to finite systems. Even if o was omniscient only over the limited applications domain of transform properties, then probability and computational error would not be required in *range*(o), since all outcomes would be certain and, therefore, accurate. Second, we define data security in terms of the information that an oracle can determine with nonzero probability. Thus, we avoid the problem of determining what an oracle does not know, which is undecidable over infinite sets, and may be undecidable over certain finite sets. Since numerous encryptions have a keyspace that is computed from the real or complex numbers, such considerations are germane, but detailed discussion is beyond the scope of this dissertation. Therefore, instead of concentrating upon oracular failure, we emphasize the more manageable problem of whether salient properties of *domain*(T) are leaked to *range*(T).

Given the following definitions of saliency, we introduce our discussion of data security with a convenient definition of security.

4.4.4.8. Definition. Per Definition 4.4.4.1, assume that an oracle $o \in \mathfrak{O}$ is applied to a transform T and outputs n properties, where each property q(i), i=1..n, has associated with it a probability p(i) and a computational error ϵ(i). A property q(i) is deemed *operationally salient* if $q(i) \in O \subset \mathcal{P}(T)$, where O denotes a predetermined set of *operationally salient properties.* Additional criteria for determining saliency may include the restriction that property q(i) in the output of oracle o must exhibit a probability p(i) that exceeds some threshold t. A similar argument can be made for the associated error measure ϵ(i).

4.4.4.9. Example. Suppose a transform T leaks the nonconstant histogram $\mathbf{h}$ of its input. If an adversary has plaintext n-gram statistics at his disposal, then *leaks* $\mathbf{h}$ is an operationally salient property.

4.4.4.10. Definition. A transform T is *secure* if and only if an oracle $o \in \mathfrak{O}$ determines that no properties of *domain*(T) are leaked to *range*(T). Given an algebra $A \in \mathcal{A}$ and a secure transform $T \in \mathcal{T}$ it follows that

$$p_1(o(A,T)) \cap \mathcal{P}(domain(T)) = \varnothing. \tag{169}$$

4.4.4.11. Remark. The preceding definition illustrates another reason why oracles cannot be omniscient. Namely, if an oracle was omniscient, then T could not hide any property of its input. As a result there would be (a) no information in the ciphertext and (b) no encryption transforms. This observation leads to a more realistic definition of data security, as follows.

4.4.4.12. Definition. A transform T is *operationally secure* if and only if the salient properties of *domain*(T) in $O \subset \mathcal{P}(T)$ are not disclosed to an oracle $o \in \mathfrak{O}$ that employs algebra A. In other words, if T is operationally secure, then $p_1(o(A,T)) \cap O = \varnothing$.

4.4.4.13. Observation. It follows that, if T is an *operationally secure* mapping, then neither $\mathcal{H}(T)$ nor $\mathcal{D}(T)$ are necessarily empty. For example, $\mathcal{D}(T)$ may contain properties of T that are not viewed as operationally salient, such as the *onto-ness* of T. Additionally, consider scenarios where enumeration-based attacks upon T are thought to be infeasible under operational constraints such as available computational bandwidth or storage. In such cases, it would be reasonable for O not to contain the property *preserves input size*.

4.4.4.14. Example. Consider a compression transform $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, whose output contains $(siz(\mathbf{F}) \cdot |\mathbf{X}|)/\mathrm{CR}$ bits. If one has an approximate knowledge of CR and a range of probable values for $siz(\mathbf{F})$, then one can guess probable values for the input size $|\mathbf{X}|$. However, $|\mathbf{X}|$ tells us nothing about the input's structure or about properties that can be computed from the input.

Each transform class exhibits properties that have various implications for data security. We thus present the following theorems that pertain to the data security of transforms in Classes 1–4 of our transformation taxonomy (reference Section 3.3). For the purposes of this discussion, we assume that $\mathbf{F}$, $\mathbf{G}$, $\mathbf{X}$, and $\mathbf{Y}$ are finite sets, and that the taxonomic class to which a given transform belongs would be known a priori to an adversary. Per our introduction to the previous section, we assume that each transform class is analyzed independent of implementational concerns that are peripheral to the transform mapping (e.g., key distribution, privilege levels, and keyspace configuration). In summary, our eventual objective is to analyze the security of each mapping T in transform classes 1–4, when T is considered only in the context of the given transform class. In this dissertation, we will provide several illustrative examples.

In the analysis of finite discrete encryptions, the length of the plaintext and the size of the plaintext alphabet provide sufficient information to determine an upper bound of complexity upon an enumerative attack.

4.4.4.15. Theorem. The following statements hold:

(i) Class 1 and Class 2 transforms are insecure, since they disclose the plaintext size in *range*(T), and

(ii) Class 1 and Class 3 transforms are insecure, since they disclose the alphabet size in *range*(T).

Proof. Assume that Class 1–3 transforms are as defined in Section 3.4. Then, (i) follows from the fact that *domain*(*choice*(*range*(T_i))) = *domain*(*choice*(*domain*(T_i))), where $i \in \{1,2\}$, and (ii) follows from the fact that *range*(*choice*(*range*(T_i))) = *range*(*choice*(*domain*(T_i))), where $i \in \{1,3\}$. □

4.4.4.16. Observation. Class 4 mappings can be compromised as follows. Given $T_4: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, $f: \mathbf{X} \rightarrow \mathbf{Y}$, and $g: \mathbf{F} \rightarrow \mathbf{G}$, let $T_4 = f \circ g$. For purposes of discussion, assume that the latter equation is known to the adversary, but f and g may not be known. Now, assume that subsets f_o and g_o of f and g, which constitute partial functions, are known. Then, a subset (or partial function) T_o of T_4 could be synthesized by composing f_o with g_o. Although one may not have an oracle that can attack T, one may be able to apply the appropriate oracle to T_o in order to disclose properties of *domain*(T_o) in *range*(T_o), since f_o and g_o are known.

Given the preceding scenario, assume that the oracle's output facilitates further guesses about T_4, i.e., helps the adversary enlarge his knowledge of f_o and g_o. In the presence of a fitness function that evaluates the content of the reconstructed transform T_4, such methods can be iterated in order to achieve a reconstruction of T_4 that may converge. Such methods are amenable to implementation in genetic algorithms [66] or relaxation algorithms [67].

We next discuss the preservation of set properties, with the image sum and histogram given as an example.

4.4.4.17. Theorem. A bijection (Class 1 transform) preserves the histogram of its input.

Proof. A bijection is an isomorphism. Therefore, the histogram of the plaintext input is discovered in the plaintext output by relabelling the histogram domain. □

4.4.4.18. Observation. The image sum is preserved by linear transforms of form $T(x) = ax + b$, due to the linearity property, as well as distributivity of multiplication over addition. However, the sum may be scaled or offset by a constant value.

4.4.4.19. Examples.

1. Let the linear transform $T(\mathbf{a}) = (3 * \mathbf{a}) + 4$. If $N = |domain(\mathbf{a})|$, then $\Sigma(T(\mathbf{a})) = \Sigma(3 * \mathbf{a}) + 4 = 3*\Sigma\mathbf{a} + 4N$.
2. Truncation of image grey levels by k bits can be viewed as a linear transform whose scale factor equals 2^{-k}.

With respect to data security, we described (in Chapter 1) how a linear homomorphism of size n could be compromised by solving a system of n linear equations.

4.4.4.20. Theorem. Spatial transforms (Class 3) preserve an ϵ-near approximation to the input sum and histogram, where $\epsilon = 0$ for continuous domains that are not truncated by the spatial transform.

Proof. This holds for continuous spatial transforms, since $range(range(T_3)) = range(domain(T_3))$. In the case of discrete transforms, T_3 may include interpolation of values, which usually renders $\Sigma(T_3(\mathbf{a}))$ an approximation to $\Sigma\mathbf{a}$□

The preceding theorems are key to the description of constraints upon cryptanalytic attack via combinatoric means, as applied to compressive or encryptive transformations. The difficulty (or ease) of inverting such transforms dictates the security of computations over encrypted sets. We discuss encryptive computation in the following section.

4.5. Feasibility of Encryptive Computation.

We begin by stating several conditions that influence the security of computations over encrypted data. Then, we present an abstract framework for cryptanalytic attack upon encryptive transforms, which is employed in the evaluations of transform security that is discussed primarily in Chapter 8.

4.5.1. Security and ϵ-Security of Homomorphisms

Since we can determine the security of a transform using an oracle, then we can extend such statements to computational systems embodying such transforms.

4.5.1.1. Recall. In Chapter 3, we defined the preservation of set properties under homomorphism. As we have shown in Section 4.5, properties that are preserved under transformation T are not limited merely to analogues of image operations, but can include plaintext length and alphabet size. If T is a homomorphism, then it is obviously correct to say that such properties are preserved under homomorphism. Furthermore, if such properties define the security of a transform, then it is reasonable to define secure homomorphisms as follows.

4.5.1.2. Definition. A *secure homomorphic computational system* is an homomorphic computational system in which the constituent homomorphism(s) is(are) secure according to Definition 4.4.4.10.

4.5.1.3. Definition. An *operationally secure homomorphic computational system* is an homomorphic computational system whose constituent homomorphism(s) is(are) secure with respect to a set of properties O, per Definition 4.4.4.12.

From the development of Chapter 3, it should be understood that the preceding definitions apply to isomorphic and ϵ-near approximations to homomorphic computational systems. Assuming that we have a secure HCS X_H, a key issue in encryptive computation

concerns the compromise of X_H's security by one or more analogues of an image operation, as discussed in the following theorem.

4.5.1.4. Theorem. If a homomorphic computational system $X_H = (\{F^X, \bigcirc\}, \{T, \Theta\}, \{G^Y, \bigcirc'\})$ is secure, then the analogous operation $\bigcirc'$ over *range*(T) does not compromise the security of T.

Proof. By definition of a secure HCS, T is assumed to be secure. For purposes or argument, assume that the definition of security given in Section 4.4.4.10 holds, i.e., no properties of *domain*(T) can be detected in *range*(T) by an oracle $o \in \mathfrak{O}$. Implied in this definition is the assumption that no other operation is as capable as the oracle. (If such was the case, then that operation would, by definition, become the oracle.) Thus, $\bigcirc'$, which operates over *range*(T), cannot produce any property of *domain*(T) in *range*($\bigcirc'$). Since $\bigcirc'$ lacks the oracle's capabilities, $\bigcirc'$ is said to have no information about *domain*(T) at its input or output. Thus, $\bigcirc'$ does not compromise the security of T. □

4.5.1.5. Remark. The preceding theorem may initially seem somewhat fanciful, since we know that no finite discrete encryption is safe from attack via enumeration of possible plaintexts. However, per Remark 4.4.4.11. It follows from inspection of Theorem 4.5.1.4 that an *insecure* homomorphism may have whatever security it possesses further compromised by operations conducted over *range*(T). With the exception of the trivial case of attack by enumeration, there are very few finite discrete homomorphisms that are not provably insecure. For example, the Fourier transform, which is often the basis of linear homomorphisms, can easily yield to attack via correlation, due to the correlation property of the FT. It follows that a transform T can succumb to an attack which is based upon one or more properties Ω of T that involve the action of preservation. It is easily verified that such cases occur especially when one can determine an analogue Ω' of Ω.

Since secure transforms appear to be scarce, but operationally secure transforms can appear to be plentiful (depending upon the mapping and the set of operationally salient properties), we prefer to couch the security of operations over encrypted data in terms of the following convention that derives from our previous discussion of operational security.

4.5.1.6. Assumption. Assume that an oracle $o \in \mathfrak{O}$ is applied to a transform T. Given a three-tuple (q,p,e) in *range*(o), assume that the probability measure p for a given property q denotes the *certainty* (in the information-theoretic sense) with which T discloses q to o's scrutiny of *range*(T).

4.5.1.7. Definition. Given an oracle $o \in \mathfrak{O}$ with an algebra A, a set of salient properties O, as well as probability and error thresholds ϵ_p and ϵ_e, a transform T is said to be *operationally ϵ-secure* if and only if, for all tuples (q,p,e) in $\mathbf{W}$, the following condition holds:

$$\mathbf{W} = \{(q, p, e) \in \mathit{range}(o(A,T)) : q \in O\,,\ p < \epsilon_p,\ \text{and}\ e < \epsilon_e\}\,. \tag{170}$$

We can use this concept of ϵ-security for various purposes, as shown in the following observation and example.

4.5.1.8. Observation. Under constraint of Assumption 4.5.1.6, it follows that 1−p denotes the *uncertainty* with which T *discloses* q to o and the *certainty* with which T *conceals* q from o. Thus, p denotes the *uncertainty* with which T conceals q, a property of *domain*(T), from o's scrutiny of *range*(T). As a result, ϵ-security can be thought of as a link between data security, information theory, and cryptanalysis.

The utility of this concept will become apparent in subsequent discussion concerning information-theoretic analysis of data security. We present the following example for purposes of illustration.

4.5.1.9. Example. Let Definition 4.5.1.7 hold. Given a probability threshold ϵ_p in (0,1] and an oracle $o \in \mathfrak{O}$ with algebra A, a mapping M is called *operationally mean ϵ-secure* if and only if

$$\frac{1}{|\mathbf{W}|} \sum_{w \in \mathbf{W}} p_2(w) < \epsilon_p , \tag{171}$$

where $\mathbf{W}$ was defined in terms of $o(A,T)$ in Equation 170. Similarly, *operational RMS ϵ-security* could be defined as

$$\left(\frac{1}{|\mathbf{W}|} \sum_{w \in \mathbf{W}} p_2(w^2) \right)^{1/2} < \epsilon_p . \tag{172}$$

Prior to the second example of a probabilistic oracle, we expand upon Observation 4.5.1.8 by advancing the following observations concerning the ϵ-security of transforms, which we express in terms of an ensemble of properties.

4.5.1.10. Observation. It can be easily verified by inspection of Definition 4.4.4.12 that a transform T which is operationally secure may not be secure in the sense of Definition 4.4.4.10. That is, one or more properties in O may be manifest in both the range and domain of T, albeit with different probabilities of existence. This difference in probabilities is due to the fact that the probability with which T conceals a given property q of *domain*(T) from an oracle's scrutiny of *range*(T) can vary with q. If an oracle is not omniscient (as we assume in this study), then its knowledge will be limited, and the oracle might not be able to deduce certain properties of a given encryption.

4.5.1.11. Example. Assume (for purposes of simplicity) that q and s are the only properties in O that are known properties of *domain*(T), and let an oracle $o \in \mathfrak{O}$ employ algebra A and accept T. For purposes of simplicity, further assume that $p_1(o(A,T))$ is a set containing probabilities q and s, whose probability of detection in *range*(T) is given by $p(q) = p_2(o(A,T)|_{\{(m,p,e):m=q\}})$, and symmetrically for s. Since q and s are known to be properties of *domain*(T), we say that $p(q) = 1$. Now, suppose that T is a homomorphism

upon which an HCS X_H is based. Let a threshold of probability be denoted by ϵ. Then, X_H would be ϵ-secure with respect to $O\backslash\{q\}$, under the scrutiny of o, if and only if $\epsilon \geq \vee(p(q), p(s))$. It is easily verified that a criterion for ϵ could be thusly derived from practical knowledge of a mapping and its properties.

We next formalize the concept of operational security for homomorphic computational systems, which generalizes all computational systems, as discussed in Chapter 3.

4.5.2. Operational Security of HCS

The definition of security criteria for homomorphic computational systems is key to our discussion of secure computation, since HCS and ICS form the basis for our theoretical discussion.

4.5.2.1. Definition. Given a set of operational properties O and an oracle $o \in \mathfrak{O}$, an *operationally ϵ-secure homomorphic computational system* $X_H = (\{S, \bigcirc\}, \{T, \Theta\}, \{S', \bigcirc'\})$ is an HCS that satisfies the following conditions:

(i) The homomorphism $T: S \rightarrow S'$ is an operationally ϵ-secure mapping, in the context of O and o; and

(ii) The operation $\bigcirc' : S' \times S' \rightarrow S'$, an analogue of the operation $\bigcirc : S \times S \rightarrow S$, is an operationally ϵ-secure mapping.

4.5.2.2. Theorem. Given a set of operational properties O and an oracle $o \in \mathfrak{O}$, an isomorphic computational system $X_I = (\{S, \bigcirc\}, \{T, \Theta\}, \{S', \bigcirc'\}, T^{-1})$ is operationally ϵ-secure if the following three conditions are satisfied:

(i) The HCS $X_H = (\{S, \bigcirc\}, \{T, \Theta\}, \{S', \bigcirc'\})$ is ϵ'-secure in the context of O and o;

(ii) The mapping $\Theta: \bigcirc \vdash\!\!\rightarrow \bigcirc'$ has an inverse $\Theta^{-1}: \bigcirc' \vdash\!\!\rightarrow \bigcirc$ and the HCS $X_H = (\{S', \bigcirc'\}, \{T^{-1}, \Theta^{-1}\}, \{S, \bigcirc\})$ is ϵ''-secure in the context of O and o; and

(iii) $\epsilon = \vee(\epsilon', \epsilon'')$.

Proof. The proof is outlined as follows. From Theorem 4.5.2.2 and Section 3.2, we see that a secure ICS can be constructed from two HCS X_H and X'_H, where X_H is based upon the homomorphism T and X'_H is based upon the inverse homomorphism T^{-1}. If the insecurity level of X_H and X'_H are denoted by ϵ' and ϵ'', respectively, then from Observation 4.5.1.8, the net *insecurity* is equal to $\epsilon = \vee(\epsilon', \epsilon'')$. The maximum function is employed since we seek to determine the maximum probability with which a property of *domain*(T) is manifested in *range*(T). □

4.5.2.3. Remark. We emphasize that the preceding comments hold primarily for finite sets or finite subsets of infinite sets. Since all properties of all sets are not computable, we have shown that it is sufficient to speak of security in terms of non-omniscient oracles that determine only those properties that are known to pertain to a given operational scenario. When deriving an ϵ-secure HCS, one must first consider the specification of the properties O that constrain the definition of security. Then, one can proceed with the design of an oracle, which (if it exists) would be a special case of a more general oracle $o \in \mathfrak{O}$. For example, if one knows that preservation of the histogram is a salient property, then one can proceed with the design of an oracle that deduces the certainty with which the plaintext histogram is preserved in ciphertext. We next discuss computational error and information loss.

CHAPTER 5
COMPUTATIONAL ERROR AND INFORMATION LOSS

Computations that employ discrete approximations of real numbers produce errors which can degrade the accuracy of operations over the range spaces of numerous transforms. In Section 5.1, we analyze error propagation in discrete computational systems. Given the error functions associated with basic arithmetic and transcendental operations, we then derive error predictors for the image algebra operations employed in this study (Section 5.2). In Section 5.3, we present a theory of error-tolerant computation and discuss the feasibility of such erroneous computations in Section 5.4.

5.1. Theory of Error Propagation in Discrete Systems

We begin our discussion of error analysis by considering the errors of roundoff and series truncation inherent in digital computation, after Chern [64]. Then, we analyze the error that accrues from the discrete implementation of arithmetic and transcendental functions.

5.1.1. Basic Error Measures.

We first establish several notational conventions in terms of an idealized computation.

5.1.1.1. Assumption. Let S be a normed space with norm $\| \ \|$. Assume that one defines a function $f : S \to S$ symbolically as $f(x) \equiv g(f,x)$, $x \in \mathbf{R}$, where the ideal implementation $g : (S \cup \{\perp\})^S \times S \to S$ accepts a partial function $f_o \subset f$ with $\perp$ denoting an undefined symbol, together with a value in S, and produces $f(x) \in S$.

5.1.1.2. Remark. The preceding formulation is abstract by design, and thus says nothing about the implementation of f. For example, f could be defined by a programmer in terms of the functional composition $g = f \circ x$, where x could denote a reflexive function that maps an element $f \in S$ onto itself.

5.1.1.3. Definition. In principle, let f have an exact solution that is denoted by f_e and an algorithmic solution f_a that could be solved exactly, with sufficient resources. Additionally, let the computation of f_a on an N-processor discrete architecture A with finite precision be denoted by f_c. For purposes of completeness, we denote the initial value f_o of $f(x)$ for some $x = x_o$.

5.1.1.4. Example. For example, given $S = \mathbf{R}$, if $f(x) = sin^2(x)$, $\forall x \in S$, then f_a could be the Taylor-series approximation to the symbolic expression f. The computation of f_a on a sequential architecture with 32–bit precision would be given by f_c.

5.1.1.5. Definition. Under constraint of Assumption 5.1.1.1 and Definition 5.1.1.3, the *accumulated roundoff error* of f when computed as f_c (as opposed to being computed on A as f_a) is given by:

$$\epsilon_r(x) = f_c(x) - f_a(x). \tag{173}$$

5.1.1.6. Definition. Under constraint of Assumption 5.1.1.1 and Definition 5.1.1.3, the *accumulated truncation error* associated with computing f with infinite precision using a finite (truncated) expression (series) f_a is given by:

$$\epsilon_t(x) = f_a(x) - f_e(x). \tag{174}$$

5.1.1.7. Definition. Under constraint of Assumption 5.1.1.1 and Definition 5.1.1.3, the *total accumulated error* of f when computed on A is given by:

$$\epsilon_{tot} = \epsilon_t + \epsilon_r = f_c - f_e. \tag{175}$$

5.1.1.8. Remark. For purposes of brevity, we express the total error ϵ_{tot} of a function f as the deviation δf. Similarly we express the error of an operand $f \in S$, where S is a normed space, by δf. In this dissertation, δf will be a symbolic entity primarily, while δf generally denotes the scalar error or error distribution associated with a scalar variable or image values. The context will be clear from the form of a given expression.

This notational convenience enables us to express the following error functions concisely.

5.1.2. Error Propagation in Arithmetic and Transcendental Functions.

We begin by assuming that the basis functions for discrete computing are the arithmetic functions of addition (+) and multiplication (·), as well as their inverses. We express the transcendental functions and their inverses (log, exp, sine, cosine) in terms of Taylor series expansions, which can be computed in terms of addition and multiplication.

5.1.2.1. Theorem. Let $f, g, h \in \mathbf{R}$, and let the nonzero sum $h = f(f,g) = f + g$, where $\delta f, \delta g \in \mathbf{R}$ denote the errors in f and g. The absolute error $|\delta h|$ in the nonzero sum $h = f + g \neq 0$ is bounded as $|\delta h| \leq |\delta f| + |\delta g|$.

Proof. The proof is given in Reference 60, and is well known. □

5.1.2.2. Theorem. Let $f, g, h \in \mathbf{R}$, and let the nonzero sum $h = f(f,g) = f + g$, where $\delta f, \delta g \in \mathbf{R}$ denote the errors in f and g. The absolute deviation $|\delta h|$ in the nonzero product $h = f \cdot g \neq 0$ is bounded as

$$|\delta h| \leq |f \cdot g| \cdot \left(\frac{|\delta f|}{f} + \frac{|\delta g|}{g} \right) + (\delta f \cdot \delta g). \tag{176}$$

The rightmost term is usually of negligible magnitude, and can be ignored.

Proof. The proof is given in Reference 60, and is well known. □

5.1.2.3. Theorem. Let $g, h \in \mathbf{R}$, and let the exponent $h = f(g) = e^g$, where $\delta g \in \mathbf{R}$ denotes the error in g. If the series expansion of f is given by the algorithm

$$f_a(g) = 1 + \sum_{i=1}^{n-1} \frac{g^i}{i!}, \tag{177}$$

where n is finite and $0! = 1$, then the absolute error $|\delta h|$ in f_a is bounded as

$$|\delta h| \leq \sum_{i=1}^{n-1} \frac{i \cdot |\delta g|}{|g|^{i+1}}. \tag{178}$$

The rightmost term $(|\delta i|/i!)$ is usually zero, since i is encoded as an exact (integer) quantity in most computers.

Proof. Assuming that the givens hold, we have from Theorems 5.1.2.1 and 5.1.2.2 that:

$$\begin{aligned} \delta\left(\sum_{i=1}^{n-1} \frac{g^i}{i!}\right) &= \sum_{i=1}^{n-1} \delta\left(\frac{g^i}{i!}\right) \\ &= \sum_{i=1}^{n} \left(\frac{\delta(g^i)}{|g|^i} - \frac{\delta(i!)}{i!}\right) \\ &= \sum_{i=1}^{n} \left(\frac{i \cdot \delta g}{|g| \cdot |g|^i} - \frac{1}{i!} \cdot \sum_{l=1}^{i} \left|\frac{\delta l}{l}\right|\right) \\ &= \sum_{i=1}^{n} \frac{i \cdot \delta g}{|g|^{i+1}}, \end{aligned} \tag{179}$$

since the error δl in an integer l equals zero. This result can be obtained from the right-hand member of the preceding equation by assuming that $\delta(i!) = 0$, then expanding $\delta(f^i)$ according to the product rule given in Theorem 5.1.2.2. □

5.1.2.4. Theorem. Let $h = f(g) = g^k$, $\forall g \in \mathbf{R}$ and $k \in \mathbf{Z}$, where $\delta g \in \mathbf{R}$ denotes the error in g. If f is computed in terms of the algorithm

$$f_a(g) = \prod_{i=1}^{k} g, \tag{180}$$

where k is assumed to be finite, then the absolute error $|\delta h|$ in f_a is bounded as:

$$|\delta h| \leq k \cdot \left|\frac{\delta g}{g}\right| \cdot g^k. \tag{181}$$

Proof. The proof follows straightforwardly from Theorem 5.1.2.2. □

5.1.2.5. Theorem. Let $h = f(g) = \ln(g)$, $\forall g, h \in \mathbf{R}$, such that $g > 0$, where $\delta g \in \mathbf{R}$ denotes the error in g. If the series expansion of f is given by the algorithm

$$f_a(g) = 2 \cdot \sum_{i=1}^{n} \frac{1}{2i-1} \cdot \left(\frac{g-1}{g+1}\right)^i, \tag{182}$$

where n is assumed to be finite, then the absolute error $|\delta h|$ in f_a is bounded as:

$$|\delta h| \leq n(n+1) \cdot \left|\frac{\delta g}{g}\right| + n \cdot \epsilon_t, \tag{183}$$

with ϵ_t denoting the total computational error in the quotient expression 1/(2i-1).

Proof. Assuming that the givens hold, we have from Theorems 5.1.2.1 and 5.1.2.2 that

$$\begin{aligned} \vee|\delta h| &= \vee\left|\delta\left(\sum_{i=1}^{n} \frac{1}{2i-1} \cdot \left(\frac{g-1}{g+1}\right)^i\right)\right| \\ &= n \cdot \epsilon_t + \sum_{i=1}^{n} \delta\left[\left(\frac{g-1}{g+1}\right)^i\right] \\ &= n \cdot \epsilon_t + \sum_{i=1}^{n} 2i \cdot \left|\frac{\delta g}{g}\right|. \end{aligned} \tag{184}$$

Since $|\delta h| \leq n \cdot \left(\epsilon_t + (n+1) \cdot \left|\frac{\delta g}{g}\right|\right)$, the theorem holds. □

5.1.2.6. Theorem. Let $h = f(g) = \sin(g)$, $\forall g \in \mathbf{R}$, where $\delta g \in \mathbf{R}$ denotes the error in g. If the series expansion of f is given by the algorithm

$$f_a(g) = \sum_{i=1}^{n} (-1)^{i-1} \frac{g^{(2i-1)}}{(2i-1)!}, \tag{185}$$

where n is assumed to be finite, then the absolute error $|\delta h|$ in f_a is bounded as

$$|\delta h| \leq n \cdot \epsilon_t + \left|\frac{\delta g}{g}\right| \cdot \sum_{i=1}^{n} (2i-1) \cdot g^{2i-1}, \tag{186}$$

with ϵ_t denoting the total error inherent in computing the quotient 1/(2i-1)!

Proof. Assuming that the givens hold, and noting that sign inversion carries no computational error, we further assume that the computation of $\frac{1}{(2i-1)!}$ incurs total error ϵ_t. Then, we have the following derivation:

$$\begin{aligned} \vee|\delta h| &= \vee\left|n \cdot \epsilon_t + \sum_{i=1}^{n} \delta\left(\frac{g^{2i-1}}{(2i-1)!}\right)\right| \\ &= n \cdot \epsilon_t + \sum_{i=1}^{n} \delta\left(g^{2i-1}\right) \\ &= n \cdot \epsilon_t + \sum_{i=1}^{n} (2i-1) \cdot \left|\frac{\delta g}{g}\right| \cdot g^{2i-1} \\ &= n \cdot \epsilon_t + \left|\frac{\delta g}{g}\right| \cdot \sum_{i=1}^{n} (2i-1) \cdot g^{2i-1}, \end{aligned} \tag{187}$$

which states the lower bound, where the inequality follows from the triangle inequality. □

The following method of computing the sine function yields greater accuracy and computational efficiency.

5.1.2.7. Theorem. Let $h = f(g) = \sin(g)$, $\forall g \in \mathbf{R}$, where $\delta g \in \mathbf{R}$ denotes the error in g. If the series expansion of f is given by the algorithm

$$f_a(g) = g \cdot \sum_{i=1}^{n} \left(1 - \frac{g^2}{\pi^2}\right), \tag{188}$$

where n is finite and the error inherent in computing π is denoted by ϵ_π, then the absolute error $|\delta h|$ in f_a is bounded as:

$$|\delta h| \leq \left|\frac{\delta g}{g}\right| + n \cdot \left(\frac{2 \cdot \left(\left|\frac{\delta g}{g}\right| + \epsilon_\pi\right)}{\left|1 - (g/\pi)^2\right|}\right). \tag{189}$$

with ϵ_t denoting the machine error inherent in computing the quotient 1/(2i-1)!.

Proof. Assuming that the givens hold, we have from Theorems 5.1.2.1 and 5.1.2.2 that:

$$\begin{aligned} \forall|\delta h| &= \forall\delta\left(g\cdot\sum_{i=1}^{n}\left(1-\frac{g^2}{\pi^2}\right)\right) \\ &= \left|\frac{\delta g}{g}\right| + \sum_{i=1}^{n}\frac{\delta\left(1-(g/\pi)^2\right)}{\left|1-(g/\pi)^2\right|} \\ &= \left|\frac{\delta g}{g}\right| + n\cdot\left(\frac{2\cdot\left(\left|\frac{\delta g}{g}\right|+\epsilon_\pi\right)}{\left|1-(g/\pi)^2\right|}\right), \end{aligned} \tag{190}$$

which restates the lower bound. The inequality follows from the triangle inequality, after the proof of the preceding theorem. □

5.1.2.8. Observation. Method 2 (Theorem 5.1.2.7) of computing the sine function exhibits an error which is linear in the fractional error $\left|\frac{\delta g}{g}\right|$, while Method 1 (Theorem 5.1.2.6) is quadratic in $\left|\frac{\delta g}{g}\right|$. If $\left|\frac{\delta g}{g}\right| < 1$ (which is usual), then Method 1 is more accurate, provided that π is computed with sufficient precision. However, if $|\delta g| > |g|$ then Method 2 yields greater precision.

5.1.2.9. Remark. Since the cosine function differs from the sine only by a phase constant, then the error in the cosine should be determined in a manner similar to that of the sine. Since proofs of the following assertions can be arrived at via inspection of Theorems 5.1.2.6 and 5.1.2.7, we state the error functions in brief form only. Let $g, h \in \mathbf{R}$ exhibit deviations $\delta g, \delta h \in \mathbf{R}$. If the cosine of g is given by

$$h = f(g) = \cos(g) \approx f_a(g) = \sum_{i=0}^{n}(-1)^i\cdot\frac{g^i}{2!}, \tag{191}$$

then the deviation in h is given by:

$$|\delta h| \le n\cdot\epsilon_t + \sum_{i=0}^{n} i\cdot\left(\left|\frac{\delta g}{g}\right|\cdot g^i\right), \tag{192}$$

where ϵ_t denotes the error in computing the quotient in Equation 191. Due to multiplication by i, $\left|\frac{\delta g}{g}\right|$ is taken $\mathbf{O}(n^2)$ times, which yields higher computational overhead for the error function than does the sine function.

Alternatively, if the cosine is given by the series expansion:

$$h = f(g) = \cos(g) \approx f_a(g) = \prod_{i=1}^{n} 1 - \cdot \frac{4g^2}{(2i-1)^2 \cdot \pi^2}, \tag{193}$$

then the deviation in h is given by:

$$|\delta h| \leq n \cdot \left(\frac{2\left(\epsilon_\pi + \left|\frac{\delta g}{g}\right|\right)}{\left|\frac{4g^2}{(2i-1)^2 \cdot \pi^2}\right|} \cdot g^i \right), \tag{194}$$

which follows directly from Theorem 5.1.2.2. As before, $\left|\frac{\delta g}{g}\right|$ is taken $\mathbf{O}(n^2)$ times.

5.1.2.10. Theorem. Let $g, h \in \mathbf{R}$ exhibit deviations $\delta g, \delta h \in \mathbf{R}$. If the inverse sine function is given by the series expansion:

$$h = f(g) = \sin^{-1}(g) \approx f_a(g) = \sum_{i=1}^{n} k_i \cdot g^{2i-1}, \tag{195}$$

then the deviation δh is given by:

$$|\delta h| = \left|\frac{\delta g}{g}\right| \cdot \sum_{i=1}^{n} (2i-1) \cdot g^{2i-1}. \tag{196}$$

Proof. The proof follows directly from Theorems 5.1.2.1, 5.1.2.2, and 5.1.2.6. □

5.1.2.11. Remark. The error in the inverse cosine expansion

$$h = f(g) = \cos^{-1}(g) \approx f_a(g) = \frac{\pi}{2} - \sum_{i=1}^{n} k_i \cdot g^{2i-1}, \tag{197}$$

is equal to the error in the inverse sine function plus ϵ_π, the error inherent in computing π. This is due to the phase difference of $\pi/2$ radians between the sine and cosine functions.

We next present theorems that describe the error in univariate and multivariate functions. The proofs, which are given in References 60 and 61, are well known.

5.1.3. Error in Univariate and Multivariate Functions.

It is useful to determine the error inherent in symbolic functional specifications, which are frequently encountered in certain programming languages (such as Lisp). We begin with an analysis of the general univariate function, then progress to multivariate functions and functional composition.

5.1.3.1. Theorem. Let a univariate function $f: \mathbf{X} \rightarrow \mathbf{F}$ be differentiable. The error inherent in $f(\mathbf{x})$ is given in terms of the deviation $\delta\mathbf{x}$ as

$$|\delta f|(\mathbf{x}) = \left|\frac{df(\mathbf{x})}{d\mathbf{x}}\right| \cdot \delta\mathbf{x}, \ \mathbf{x} \in \mathbf{X}, \tag{198}$$

where we assume that $\delta\mathbf{x}$ is taken in the same sense as $d\mathbf{x}$.

5.1.3.2. Theorem. Let a multivariate function $f: \mathbf{X}^m \rightarrow \mathbf{F}$ be differentiable. The error inherent in $f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$, where $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \mathbf{X}^m$, is given by

$$|\delta f| = \left(\sum_{i=1}^{m} \left|\frac{df}{d\mathbf{x}} \cdot \delta\mathbf{x}_i\right|\right)^{\frac{1}{2}}, \tag{199}$$

if the following conditions are satisfied:

(i) the errors $\delta\mathbf{x}_1, \delta\mathbf{x}_2, \ldots, \delta\mathbf{x}_m \in \mathbf{R}$ in $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \mathbf{R}$ are independent and random; and

(ii) δf is bounded as $|\delta f| \leq \sum_{i=1}^{m} \left|\frac{\partial f}{\partial \mathbf{x}_i} \cdot \delta\mathbf{x}_i\right|$.

The following theorem exhibits an obvious proof, and is a useful introduction to the discussion of error propagation in computational systems.

5.1.3.3. Theorem. Let the composition of n functions of form $g_i: S \rightarrow S$, where i = 1..n, be given by

$$h = f(g_1, g_2, \ldots, g_n) = \mathop{\circ}_{i=1}^{n} g_i, \tag{200}$$

The computational error is given by

$$\delta h = f(\epsilon_{g_1}, \epsilon_{g_2}, \ldots, \epsilon_{g_n}) = \left(\mathop{\circ}_{i=1}^{n-1} \epsilon_{g_i}\right) \circ \epsilon_{g_n}(\delta g), \tag{201}$$

where ϵ_{g_i} denotes the error function of g_i and the deviation δg denotes the input error.

We thus have the basis for determining error propagation in image algebra operations, as discussed in the following section.

5.2. Error Propagation in Discrete Image Algebra Operations

Since the algorithms of this study are expressed in terms of image algebra, we present the following analysis of error propagation through discrete IA operations. Due to the fact that pointwise image operations obey the error propagation rules stated in Section 5.1, we begin with the global reduce functions of image sum, product, and maximum. We then analyze error propagation through the floor and ceiling functions, as well as the image histogram, median, and moments (central and noncentral). We conclude with a brief discussion of errors inherent in computing the image-template product.

Unless otherwise specified, we make the usual assumption that the specification of pixel locations in the image domain $\mathbf{X}$ is exact. Thus, we adopt the convention that errors occur as a result of perturbations in image pixel values.

5.2.1. Global Reduce Function

We begin our discussion of error propagation in global reduce function with the common case of image summation. It is easily verified that the mean error in the image sum is bounded above by the sum of the mean pixel errors, as shown in the following theorem.

5.2.1.1. Theorem. Let image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ and let the absolute mean deviation per pixel be denoted by $\delta \mathrm{r} = \frac{1}{|\mathbf{X}|} \cdot \sum_{\mathbf{x} \in \mathbf{X}} |\delta(\mathbf{a}(\mathbf{x}))|$. Then, the mean error $\overline{\delta \mathrm{h}} \in \mathbf{R}$ that results from the image sum $\mathrm{h} = f(\mathbf{a}) = \Sigma \mathbf{a}$ is bounded as $|\overline{\delta \mathrm{h}}| \leq |\mathbf{X}| \cdot \delta \mathrm{r}$.

Proof. Assuming that the givens hold, then from Theorem 5.1.2.1, we have that

$$|\overline{\delta \mathrm{h}}| = \sum_{\mathbf{x} \in \mathbf{X}} \delta(\mathbf{a}(\mathbf{x})) = |\mathbf{X}| \cdot \left(\frac{1}{|\mathbf{X}|} \cdot \sum_{\mathbf{x} \in \mathbf{X}} \delta(\mathbf{a}(\mathbf{x})) \right) \leq |\mathbf{X}| \cdot \delta \mathrm{r}, \tag{202}$$

where the inequality follows from the triangle inequality. □

5.2.1.2. Example. Let $\mathbf{a} \in (\mathbf{Z}_{2^{16}})^{\mathbf{X}}$ and let δr represent an error of ± 1 least significant bit (LSB), i.e., $|\delta r| = 1$. Assuming uniformly distributed image values, the average value of each pixel will equal $|\mathbf{X}| \cdot 2^{16}/2 = 2^{15}$. If $|\mathbf{X}| = 1\text{M}$ pixels, then $\overline{\Sigma \mathbf{a}} = |\mathbf{X}| \cdot (2^{15}) = 2^{20} \cdot 2^{15} = 2^{35} \approx 3.43 \times 10^{10}$. Thus, the mean error per pixel in the sum is bounded as $\overline{\delta(\Sigma \mathbf{a})} \leq \Sigma \mathbf{a}/|\mathbf{X}| \approx \frac{3.43 \times 10^{10}}{1.05 \times 10^{6}} = 3.26 \times 10^{4}$.

5.2.1.3. Theorem. Let image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ and let the mean deviation per pixel be denoted by $\delta r \in \mathbf{R}$. If the image product $h = f(\mathbf{a}) = \prod \mathbf{a}$, then the error δh in h is bounded as:

$$\delta h \leq \prod (\mathbf{a}) \cdot \sum_{\mathbf{x} \in \mathbf{X}} \frac{\delta r}{|\mathbf{a}(\mathbf{x})|}. \tag{203}$$

Proof. The result follows directly from Theorem 5.1.2.2. □

From the preceding two theorems, it is obvious that the pixelwise error is a significant measure. In particular, it is useful to express the expected error in a space-variant fashion, since many imaging devices exhibit spatially variant SNR and PSF effects. Additionally, the following convention can simplify the formulation of error propagation equations.

5.2.1.4. Definition. An *error image* $\mathbf{e} \in \mathbf{R}^{\mathbf{X}}$ of an image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ is defined as $\mathbf{e}(\mathbf{x}) = \delta(\mathbf{a}(\mathbf{x}))$, $\mathbf{x} \in \mathbf{X}$.

5.2.1.5. Example. From Theorem 5.1.2.1, the error in the image sum is given by $|\delta(\Sigma \mathbf{a})| \leq \Sigma \mathbf{e}$. From Theorem 5.1.2.2, the error in the image product $|\delta(\prod \mathbf{a})| \leq \Sigma(\mathbf{e}/\mathbf{a}) \cdot \prod \mathbf{a}$.

We conclude our discussion of the global reduce function with the image maximum $\vee \mathbf{a} = \bigvee_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x})$, which can be analyzed in terms of the binary function $\vee : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$.

5.2.1.6. Theorem. Let $f, g, h \in \mathbf{R}$, with respective deviations $\delta f, \delta g, \delta h \in \mathbf{R}$. If the maximum is expressed in terms of addition, i.e.,

$$\begin{aligned} h &= f(f, g) = \vee(f, g) \\ &\approx f_{\mathrm{a}}(f, g) = \begin{cases} f & \text{if } (f - g) \geq 0 \\ g & \text{otherwise} \end{cases}, \end{aligned} \tag{204}$$

then the error δh in h is bounded as $|\delta h| \leq |\delta f| + |\delta g|$.

Proof. The proof follows directly from Theorem 5.1.2.1, since the operation of maximum is defined in terms of addition. □

5.2.1.7. Observation. It follows from the preceding theorem that the global reduce function of maximum has an error function that is identical to that of the image sum, since both functions are expressed in terms of addition.

5.2.2. Additional Image Functions

We now progress to the analysis of the nonlinear operations of floor and ceiling, then derive the errors inherent in higher-level image functions, such as the histogram, median, and mean.

5.2.2.1. Theorem. Let $g, h \in \mathbf{R}$ exhibit error $\delta g, \delta h \in \mathbf{R}$. If the ceiling function is computed exactly as $h = f(g) = \lceil g \rceil$ and is implemented in terms of the algorithm

$$h \approx f_a(g) = i \in \mathbf{Z} \text{ if } i - 1 < g \leq i, \tag{205}$$

then the error δh is bounded as

$$|\delta h| < |\delta g| + 1 \tag{206}$$

Proof. Assume that the givens hold. Since the ceiling function quantizes g into one extremum of the interval (i-1,i], the maximum error in the quantization f_a is less than 1. If we define the ceiling function in terms of addition, then we have from Theorem 5.1.2.1 that $|\delta h| < |\delta g| + 1$. □

5.2.2.2. Remark. It is easily verified that the floor function, which is defined symmetrically to the ceiling function, exhibits an error function that is identical to that of the ceiling function. Similarly, rounding of a real number g to the nearest integer exhibits error $|\delta h| \leq |\delta g| + 0.5$, which is due to the quantization error being less than or equal to 0.5.

5.2.2.3. Theorem. Let $\mathbf{F}$ denote a finite subset of $\mathbf{R}$ and $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$. Denote the histogram $\mathbf{h} \in \mathbf{N}^{\mathbf{F}}$, such that $\mathbf{h}(r) = \Sigma\, \chi_{=r}(\mathbf{a})$, where $r \in \mathit{range}(\mathbf{a})$. If $\mathbf{e} \in \mathbf{R}^{\mathbf{X}}$ is the absolute error image of $\mathbf{a}$, and the values in $\mathit{range}(\mathbf{a})$ are sorted in ascending order as $(r_1, r_2, \ldots, r_i, \ldots)$, then the counting error specific to the i-th histogram bin is bounded as:

$$\delta(\mathbf{h}(r_i)) \leq \Sigma \vee \left(\chi_{\geq s(i,0)}(\mathbf{e}),\, \chi_{\geq s(i,-1)}(\mathbf{e})\right), \tag{207}$$

where $s(i,j) = r_{i+1+j} - r_{i+j}$.

Proof. The proof is outlined as follows. Assuming that the givens hold, let $r_{i-1}, r_i, r_{i+1} \in \mathbf{R}$ denote pixel values in image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ that are adjacent domain points of the histogram $\mathbf{h} \in \mathbf{N}^{\mathbf{F}}$. Assume that histogram bin boundaries are situated midway between each pair of adjacent values in $\mathbf{F}$, as is usual. We have the following two cases:

Case 1: If $\mathbf{e}(\mathbf{x}) = |\delta(\mathbf{a}(\mathbf{x}))| > r_{i+1} - r_i$ and $\mathbf{a}(\mathbf{x}) = r_i + |\mathbf{e}(\mathbf{x})|$, then $\mathbf{a}(\mathbf{x})$ would not be classified in the i-th bin of $\mathbf{h}$, but would be classified in the j-th bin, where $j > i$. The counting error would thus be less than or equal to $\Sigma\, \chi_{\geq s(i,-1)}(\mathbf{e})$.

Case 2: If $\mathbf{e}(\mathbf{x}) = |\delta(\mathbf{a}(\mathbf{x}))| > r_i - r_{i-1}$ and $\mathbf{a}(\mathbf{x}) = r_i - |\mathbf{e}(\mathbf{x})|$, then $\mathbf{a}(\mathbf{x})$ would be misclassified in the j-th bin of $\mathbf{h}$, where $j < i$. The counting error would thus equal $\Sigma\, \chi_{\geq s(i,-1)}(\mathbf{e})$.

We use the relation ($\geq$) together with the characteristic function, since $\mathbf{e}$ denotes the absolute error. If the error was not absolute, then the second case would pertain to negatively-signed error. Additionally, in order to provide an upper bound, we use the maximum function. It is not known a priori if Case 1 or Case 2 holds for each pixel, so we would compute both in practice, and employ the worst case as an error bound. □

5.2.2.4. Theorem. Let $\mathbf{F}$ denote a finite subset of $\mathbf{R}$, let $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, and let the histogram $\mathbf{h} \in \mathbf{N}^{\mathbf{F}}$ have an error image $\mathbf{e} \in \mathbf{R}^{\mathbf{F}}$ and a cumulative histogram $\mathbf{c} \in \mathbf{N}^{\mathbf{F}}$. Assume that values in *range*($\mathbf{a}$) are sorted in ascending order as $r_1, r_2, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_n$. If the *median* $q \in \mathbf{R}$ of $\mathbf{a}$ is defined as:

$$q = f(\mathbf{a}) = \wedge\, domain(\|\mathbf{h}\|_{>\Sigma\mathbf{h}/2}), \tag{208}$$

then the error δq in computing the median is bounded as:

$$\delta q \leq |r_i - r_j| \quad \text{where} \quad \mathbf{c}(r_j) - \mathbf{e}(r_j) \leq \frac{|\mathbf{X}|}{2} \leq \mathbf{c}(r_j) + \mathbf{e}(r_j). \tag{209}$$

Proof. The proof is outlined as follows. Assume that the givens hold and note that the computation of $\mathbf{c}$ is an integer operation, which incurs no error. However, if the histogram exhibits error $\mathbf{e}$, then error can accrue in the cumulative histogram. In particular, $\mathbf{e}$ cannot be absolute, since $\Sigma\mathbf{h} + \Sigma\mathbf{e}$ would exceed $|\mathbf{X}|$. If $\bar{e}$ denotes the average (expected) error in a given bin of $\mathbf{h}$, then we have two cases:

Case 1: If $\mathbf{c}(r_i) - \bar{e} \leq \frac{|\mathbf{X}|}{2} \leq \mathbf{c}(r_i) + \bar{e}$, then $q = r_i$ typically, and there is no counting error at the i-th bin.

Case 2: Else, $\mathbf{c}(r_j) \leq \frac{|\mathbf{X}|}{2} \leq \mathbf{c}(r_{j+1})$ and there is an error that equals $|r_i - r_j|$.

Thus, the theorem holds. □

5.2.2.5. Theorem. Let $\mathbf{X}$ denote an m×n-pixel array of points, and let $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$. Let the noncentral moment m_{pq} of order pq be defined in IA as:

$$m_{pq}(\mathbf{a}) = \Sigma(\mathbf{i}^{\mathbf{p}} * \mathbf{j}^{\mathbf{q}} * \mathbf{a}), \tag{210}$$

where $\mathbf{i} \equiv \{(\mathbf{x}, \mathbf{i}(\mathbf{x})) : \mathbf{i}(\mathbf{x}) = p_1(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}$ and $\mathbf{j} \equiv \{(\mathbf{x}, \mathbf{j}(\mathbf{x})) : \mathbf{j}(\mathbf{x}) = p_2(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}$. The error function of the noncentral moment is given by:

$$\delta(m_{pq}(\mathbf{a})) = \delta(\Sigma\mathbf{a}), \tag{211}$$

where $\delta(\Sigma\mathbf{a})$ was specified in Theorem 5.1.2.1.

Proof. Assume that the givens hold, and that $\mathbf{i}$ and $\mathbf{j}$ exhibit no error. Since neither $\mathbf{i}^p * \mathbf{j}^q$ or its summation exhibit error, Equation 210 indicates that the error in m_{pq} is given by $\delta(\Sigma\mathbf{a})$, which is described in Theorem 5.1.2.1. □

5.2.2.6. Theorem. Let $\mathbf{X}$ denote an m×n-pixel array of points and $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$. Letting $\mathbf{i} \equiv \{(\mathbf{x}, \mathbf{i}(\mathbf{x})) : \mathbf{i}(\mathbf{x}) = p_1(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}$ and $\mathbf{j} \equiv \{(\mathbf{x}, \mathbf{j}(\mathbf{x})) : \mathbf{j}(\mathbf{x}) = p_2(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}$, the mean images $\bar{\mathbf{i}}$ and $\bar{\mathbf{j}}$ defined as

$$\bar{\mathbf{i}} = (\mathbf{i} * \mathbf{a})/\Sigma\mathbf{a} \qquad \text{and} \qquad \bar{\mathbf{j}} = (\mathbf{j} * \mathbf{a})/\Sigma\mathbf{a}. \tag{212}$$

Let the nonzero central moment μ_{pq} of order pq be given by

$$\mu_{pq}(\mathbf{a}) = \Sigma\left((\mathbf{i}-\bar{\mathbf{i}})^p * (\mathbf{j}-\bar{\mathbf{j}})^q * \mathbf{a}\right). \tag{213}$$

If $\mathbf{e} \in \mathbf{R}^{\mathbf{X}}$ denotes the error image of $\mathbf{a}$, and $\mathbf{e}$ is nowhere zero-valued, then the error in μ_{pq} is bounded as

$$|\delta(\mu_{pq}(\mathbf{a}))| \leq ((p+q+1) \cdot \vee(\mathbf{e}/\mathbf{a}) + (p+q) \cdot (\Sigma\mathbf{e}/\Sigma\mathbf{a})) \cdot \mu_{pq}(\mathbf{a}), \tag{214}$$

and the average error is given by

$$\bar{\delta}(\mu_{pq}(\mathbf{a})) \leq \mu_{pq}(\mathbf{a}) \cdot (\overline{\mathbf{e}_\mathbf{a}} + (p+q) \cdot (\Sigma\mathbf{e}/\Sigma\mathbf{a})), \tag{215}$$

where $\overline{\mathbf{e}_\mathbf{a}} = \Sigma(\mathbf{e}/(\mathbf{a} * |\mathbf{X}|))$.

Proof. The proof is outlined as follows. Assume that the givens hold and assume that $\mathbf{i}$ and $\mathbf{j}$ are not erroneous. Then, we have

$$\delta(\bar{\mathbf{i}}) = \delta\left(\frac{\mathbf{i} * \mathbf{a}}{\Sigma\mathbf{a}}\right) = \frac{\delta\mathbf{a}}{\mathbf{a}} + \frac{\delta(\Sigma\mathbf{a})}{\Sigma\mathbf{a}}, \tag{216}$$

and $\delta(\bar{\mathbf{j}})$ is specified symmetrically. Thus,

$$\delta(\mu_{pq}(\mathbf{a})) \leq \mu_{pq}(\mathbf{a}) \cdot \left(p \cdot \frac{\delta\bar{\mathbf{i}}}{|\bar{\mathbf{i}}|} + q \cdot \frac{\delta\bar{\mathbf{j}}}{|\bar{\mathbf{j}}|} + \frac{\delta\mathbf{a}}{|\mathbf{a}|}\right). \tag{217}$$

Let $\mathbf{e} = \delta\mathbf{a}$ denote the error image of $\mathbf{a}$. By substitution of Equation 216 into Equation 217, we have Equation 214. □

The average error is proven symmetrically.

5.2.3. Image-Template Product.

The image-template operation $\textcircled{?}$ is the most powerful operation in IA, and has the greatest potential for error propagation, since $\textcircled{?}$ computes over image and template values that may be erroneous. We begin by analyzing the error in the common case of $\textcircled{?} = \oplus$, then discuss higher-level methods for error analysis of $\textcircled{?}$.

5.2.3.1. Theorem. Let $\mathbf{X}$ denote an m×n-pixel array of points, and let $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$. Let template $\mathbf{t} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{Y}}$ with $\mathbf{e}, \mathbf{f} \in \mathbf{R}^{\mathbf{X}}$ denoting the error images of $\mathbf{a}$ and $\mathbf{t}$, respectively, such that

$$\mathbf{f}(\mathbf{y}) = \sum_{\mathbf{x} \in \delta(\mathbf{t}_\mathbf{y})} \delta(\mathbf{t}_\mathbf{y}(\mathbf{x})), \mathbf{y} \in \mathbf{Y}, \tag{218}$$

where absolute error is assumed. If $\mathbf{c} = \mathbf{a} \oplus \mathbf{t}$, then the error $\delta\mathbf{c}$ in the result is bounded as:

$$|\delta(\mathbf{c}(\mathbf{y}))| \leq \left(\sum \frac{\mathbf{e}}{\mathbf{a}}\right) + \frac{\mathbf{f}(\mathbf{y})}{\sum_{\mathbf{x} \in \delta(\mathbf{t}_\mathbf{y})} |\mathbf{t}_\mathbf{y}(\mathbf{x})|}, \mathbf{y} \in \mathbf{Y}. \tag{219}$$

Proof. Assuming that the givens hold, a pixel value $\mathbf{c}(\mathbf{y})$ in the convolved image $\mathbf{c} = \mathbf{a} \oplus \mathbf{t}$ is given by:

$$\mathbf{c}(\mathbf{y}) = \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x}) \cdot \mathbf{t}_\mathbf{y}(\mathbf{x}), \mathbf{y} \in \mathbf{Y}. \tag{220}$$

The error $\delta(\mathbf{c}(\mathbf{y}))$ is derived as

$$\begin{aligned} \delta(\mathbf{c}(\mathbf{y})) &= \delta\left(\sum_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x}) \cdot \mathbf{t}_\mathbf{y}(\mathbf{x})\right) \\ &= \sum_{\mathbf{x} \in \mathbf{X}} \delta(\mathbf{a}(\mathbf{x}) \cdot \mathbf{t}_\mathbf{y}(\mathbf{x})) \\ &= \sum_{\mathbf{x} \in \mathbf{X}} \left(\frac{\delta(\mathbf{a}(\mathbf{x}))}{|\mathbf{a}(\mathbf{x})|} + \frac{\delta(\mathbf{t}_\mathbf{y}(\mathbf{x}))}{|\mathbf{t}_\mathbf{y}(\mathbf{x})|}\right) \\ &= \sum_{\mathbf{x} \in \mathbf{X}} \left(\frac{\mathbf{e}(\mathbf{x})}{\mathbf{a}(\mathbf{x})}\right) + \frac{\mathbf{f}(\mathbf{y})}{\sum_{\mathbf{x} \in \mathbf{X}} |\mathbf{t}_\mathbf{y}(\mathbf{x})|}, \end{aligned} \tag{221}$$

which restates the theorem. □

The error propagation function for the right product $\mathbf{c} = \mathbf{t} \oplus \mathbf{a}$ is symmetrically defined.

5.2.3.2. Remark. The error in the generalized image-template product $\textcircled{?}$ is computed similarly to the method of Theorem 5.2.3.1. Assuming that $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$, we let $\mathbf{c}(\mathbf{y}) = \underset{\mathbf{x}\in\mathbf{X}}{\Gamma} \mathbf{a}(\mathbf{x}) \bigcirc \mathbf{t}_{\mathbf{y}}(\mathbf{x})$, $\mathbf{y} \in \mathbf{Y}$, where $\bigcirc, \gamma : \mathbf{F} \times \mathbf{F} \to \mathbf{F}$ denote associative, commutative functions. We compute the deviation in $\mathbf{c}(\mathbf{y})$ by first applying the error propagation function for γ. For example, if $\gamma = +$, then the law of sums (Theorem 5.1.2.1) applies. We next apply the error function specific to the operation $\bigcirc$. For example, if $\bigcirc = \cdot$, then the law of products (Theorem 5.1.2.2) is applied. Since we have defined error propagation in Section 5.1 for all arithmetic operations that are presently employed in IA image-template products, the construction of the composite error function is primarily a matter of algebraic manipulation. As before, the error functions for the right and left products are symmetrically defined.

5.3. Theory of Error-Tolerant Computation

In Chapter 4, we discussed the optimization of computational networks comprised of operations and transforms, which optimization was based upon the criterion of time complexity. In this section, we discuss a similar situation, namely, optimizations based upon a criterion of computational error that results from a given image processing algorithm.

5.3.1. Basic Concepts.

5.3.1.1. Observation. Recall, from Theorem 5.1.3.3, that composition of functions yields an isomorphic composition of error functions. That is, if $g = f_1 \circ f_2 \circ \cdots \circ f_n$ and f_i denotes the i-th error function, then the net error function is given by $\epsilon_g = \left(\underset{i=1}{\overset{n-1}{\circ}} \epsilon_{f_i} \right) \circ \epsilon_{f_n}(\delta x)$, where δx symbolizes the input error.

5.3.1.2. Assumption. Let the approximate isomorphic computational system $X = (\{S, \bigcirc\}, \{T, \Theta\}, \{S', \bigcirc'\}, T^*)$, where the homomorphism $T: S \rightarrow S'$ has an ϵ-near approximate inverse $T^*: S' \rightarrow S$. Given that $\bigcirc(f,g) \approx T^*(\bigcirc'(T(f), T(g))), \forall f, g \in S$, denote the error function associated with $\bigcirc$ as $\epsilon_{\bigcirc}(\delta f, \delta g)$, and likewise denote ϵ_T, ϵ_{T^*}, and $\epsilon_{\bigcirc'}$. Further assume that computation of $T^*(\bigcirc'(T(f), T(g)))$ is more efficient for all $f, g \in S$, than the computation of $\bigcirc(f,g)$.

5.3.1.3. Observation. Under constraint of Assumption 5.3.1.2, if n denotes the input size, $n' = n/CR_d$, and we assume that $n'' < n$, then from Section 4.4, the work

$$W_{\bigcirc}(n) > W_T(n) + W_{\bigcirc'}(n') + W_{T^*}(n''). \tag{222}$$

Similarly, if $\epsilon_{\bigcirc} > \epsilon_{T^*} \circ \epsilon_{\bigcirc'} \circ \epsilon_T$, then (under constraint of reasonable work and error bounds), the computation $T^*(\bigcirc'(T(f), T(g)))$ may be less accurate than computing $\bigcirc$.

Assume that a given function f exhibits time complexity T_f, space complexity S_f, and has an error function ϵ_f. In order to design a computational system that implements f under the aforementioned constraints, one would employ the usual *time-space-error bandwidth product*, denoted by $\nu_f = (T_f \cdot S_f \cdot \epsilon_f)^{-1}$. Alternatively, a method of stochastic search, such as genetic algorithms, might be used to find the computationally optimal solution to the graph shown in Figure 4. In practice, the function ν, T, W, S, or ϵ can be substituted for the time complexity in the equations of Section 4.4. Since the dataflow graph of the performance function ν is isomorphic to the dataflow graph of the computational system that ν represents, the optimization theory given in Chapter 4 can be applied straightforwardly to error optimization.

5.3.2. Multivariate Error Functions.

In the analysis of error, one encounters the realistic case of multivariate functions, whose errors are modelled as follows.

5.3.2.1. Theorem. Under constraint of Assumptions 5.3.1.2 and Observation 5.3.1.3, if $\bigcirc$ and $\bigcirc'$ are m-ary functions over S and S′, respectively, with arguments $f_1, f_2, \ldots, f_i, \ldots, f_m \in S$, then the computation of the right member of

$$\bigcirc(f_1, f_2, \ldots, \ldots f_i, \ldots, f_m) \approx T^*(\bigcirc'(T(f_1), T(f_2), \ldots, T(f_i), \ldots, T(f_m))) \tag{223}$$

incurs the error

$$|\epsilon_{\bigcirc}| \leq \left| \epsilon_{T^*} \circ \left(\sum_{i=1}^{m} \left| \frac{d\bigcirc'}{dT/\delta(f_i)} \cdot \epsilon_T(\delta f_i) \right|^2 \right)^{1/2} \right|, \tag{224}$$

where δf_i denotes the deviation in f_i.

Proof. Assuming that $d\bigcirc'/df_i'$ is to be taken, and that $df_i' = dT(f_i)/df_i$, the proof follows directly from Theorems 5.1.2.1 and 5.1.2.2, and the chain rule of derivatives. Note that the error function for T^* may require additional error parameters than the one provided in Equation 224. □

5.3.2.2. Observation. In a system that has more than one operation $\bigcirc$ and more than one compressive format, the error function ϵ_T would be replaced by the error inherent in the conversion function $C_{kl}(\mathbf{a}_c) \approx T_l(T_k^*(\mathbf{a}_c))$, which we denote as $\epsilon_{C_{kl}}$.

As a result of the preceding theory, we can predict the error inherent in a computational system. The optimization theory of Chapter 4 can be applied to Equation 224, yielding an error-optimized computation. We next discuss practical aspects of error optimization, as well as implementational issues of error-tolerant computation.

5.4. Feasibility of Error-Tolerant Computation

In the previous section, we derived error functions for approximate isomorphic computational systems that are based on transforms that have ϵ-near approximations to their inverses. Additionally, we showed that the temporally-constrained optimization described

in Section 4.4 can be applied to error optimization. In this section, we discuss the feasibility of error detection and measurement, as well as the utility of such methods in image processing.

5.4.1. Automatic Error Profiling.

It is easily verified that the error functions described in Section 5.1 and 5.2 could be incorporated in computer software, such as a compiler. During execution, the accumulated error would be computed and subsequently reported to the algorithm designer or user. Such software has been designed and implemented in prototype form by the author [62]. The advantages of this technique, called *error profiling*, include

1. Interactive error display in interpretive software can aid algorithm development via selection of functions that yield near-optimal error figures;
2. The prediction of accrued error for given input data would greatly aid the performance and reliability analysis of algorithms employed in field scenarios, such as automated target recognition applications; and
3. Error profiling would furnish useful performance data for the validation of computational accuracy in processors and compilers, which is currently lacking in many software testing efforts.

An example of item 1 can be found in the development of medical image processing software. In such circumstances, a physician using an image enhancement package may want to know the amount of information loss encountered at various spatial frequencies. Such reports are key to the determination of image feature detectability, as in cases of incipient tumors, early signs of vascular aneurysm, and the microhemorrhages commonly manifested in diabetic retinopathy [63]. Via theory given in the following section, one can determine the effect of a given error in terms of information-theoretic measures such as entropy and uncertainty. We present several related applications in Chapters 6–10.

In the field of cryptology, the analysis of certain message formats can be fraught with error when the encryption transform operates near the machine's error limit. Similarly, the analysis of airborne or spaceborne reconnaissance imagery requires knowledge of the error produced by image enhancement procedures. In particular, the derivation of lexical or semantic information from data processed with such inverse transforms may be erroneous, due to errors inherent in the transform inversion process, such as division by small magnitudes in the inverse Fourier transform. Per item b), above, the design of cryptanalytic or enhancement procedures could be greatly aided by error prediction at each function stage.

However, the computational work incurred by the previously-given error functions can exceed the cost of computing the function whose error is being estimated. For example, estimation of the product error ϵ_X (Theorem 5.1.2.2) requires two divisions, two multiplications, and one addition operation. In this case, $W_{\epsilon_X} > 4 \cdot W_X$. Due to their high cost, such computations would likely be useful primarily for software design and testing.

An open research question pertains to the errors inherent in digital image acquisition. Although sampling and quantization errors have been extensively analyzed, published reports have infrequently focused on the comparison of focal-plane irradiance with the digital output of the image acquisition hardware and software. Unfortunately, focal-plane radiometry, which would be used to measure detector input, is a difficult procedure in a research environment and more so in the field. Thus, it is appropriate to consider the supporting technologies that may be required for accurate quantification of device errors in support of error profiling. Given the significance of input errors in the formulae of Sections 6.1 and 6.2, such analyses should be available to the algorithm designer in a form that can be easily manipulated. Thus, much work remains in the development of accurate, user-friendly calibration and error measurement techniques for digital imaging.

The remainder of this section contains a summary of error measures, input errors, and examples of propagation errors. We emphasize the representation of error in terms of a joint magnitude and error distribution. Via such representations, we can support

accurate computation of the error functions given in Sections 5.1 and 5.2 when the error distribution is computed over an entire image.

5.4.2. Joint Magnitude/Error Distributions.

We begin with a summary of error measures, then define the joint magnitude-error distribution (JMED) of an image. Subsequent remarks pertain to the use of JMED in the analysis of error propagation in discrete systems.

5.4.2.1. Definition. For purposes of discussion, let a source image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ be derived from a hypothetical image acquisition device that yields an error image $\mathbf{e} \in \mathbf{R}^{\mathbf{X}}$. Salient error measures can be computed from **e**, as follows:

- Sum-squared error $\epsilon_{ss}(\mathbf{a}) = \Sigma \mathbf{e}^2$
- Average or mean error $\bar{\epsilon}(\mathbf{a}) = \Sigma \mathbf{e} / |\mathbf{X}|$
- RMS error $\epsilon_{rms}(\mathbf{a}) = \left(\Sigma \mathbf{e}^2 / |\mathbf{X}|\right)^{1/2}$
- Error histogram $\mathbf{h}(\mathbf{e}) = \sum_{\mathbf{r} \in range(\mathbf{e})} \chi_{=\mathbf{r}}(\mathbf{e})$

5.4.2.2. Definition. Let the *joint magnitude-error distribution* (JMED) of image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ be denoted by $e(\mathbf{a}) \in \mathbf{R}^{\mathbf{U} \times \mathbf{V}}$, where $\mathbf{U} = f(range(\mathbf{a}))$, $\mathbf{V} = h(range(\mathbf{e}))$, with f and h denoting indexing functions. Let f and h have inverses f^{-1} and h^{-1}, such that $e(i,j)$ represents the probability of a value $f^{-1}(i)$ whose error measure is given by $h^{-1}(j)$.

5.4.2.3. Remark. It is easily verified that the functions f and h are practical for values in a finite mapping based upon $\mathbf{Z}_m$, where m is small. For example, assume the normal case of m = 256. Under the assumption of a maximum error of ±100 percent in any given value, h would range over the interval [-256, +512]. Thus, the space complexity of e would be m(3m+1), or $\mathbf{O}(M^2)$, which is not burdensome for m = 256. Now suppose that the error distributions contained in the rows of e could be expressed in terms of fractional error. Thus,

if a sampling interval exceeding 1/2m was employed as the increment for indexing $\mathbf{V}$, then the space requirement specific to the second coordinate of *domain*(e) would be reduced.

We previously (Section 5.3) alluded to errors in image acquisition, which are assumed to consist mainly of pixel magnitude errors and quantization errors, described as follows.

5.4.2.4. Remark. Let a camera project irradiance reflected or emitted from a scene comprised of three-dimensional objects to an m×n-pixel focal plane $\mathbf{X}$, thus forming an image $\mathbf{a} \in (\mathbf{Z}_m)^{\mathbf{X}}$. We assume a magnitude quantization error of ±1LSB, which will equal $\pm 1/m$. Suppose that the camera exhibits an error distribution of $\Pr(\epsilon_r)$, where ϵ denotes the error associated with a given value $r \in$ *range*($\mathbf{a}$). Then, from Definition 5.4.2.2, the JMED of $\mathbf{a}$ is given by:

$$e(f(r),\, h(z)) = \Pr(\epsilon_r), r \in \mathit{range}(\mathbf{a}), z \in \mathit{range}(\epsilon_r), \tag{225}$$

which implies that $\epsilon_r \in \mathbf{R}^{\mathbf{V}}$.

The JMED is a convenient construct for analyzing error propagation in discrete systems, as follows.

5.4.2.5. Definition. Let images $\mathbf{a},\mathbf{b} \in (\mathbf{Z}_m)^{\mathbf{X}}$ exhibit the corresponding JMED image $e(\mathbf{a}), e(\mathbf{b}) \in \mathbf{R}^{\mathbf{U}\times\mathbf{V}}$, where $\mathbf{U} = \mathbf{Z}_m$ and $\mathbf{V}$ is comprised of error fractions on the real interval [0,1]. If the scalar operation $\bigcirc$: $\mathbf{Z}_m \times \mathbf{Z}_m \rightarrow \mathbf{Z}_m$ exhibits the *JMED error function* $\epsilon_{\bigcirc}$: $\mathbf{R}^{\mathbf{U}\times\mathbf{V}} \times \mathbf{R}^{\mathbf{U}\times\mathbf{V}} \times (\mathbf{Z}_m)^{\mathbf{X}} \rightarrow \mathbf{R}^{\mathbf{U}\times\mathbf{V}}$, then assuming that $\mathbf{c} = \mathbf{a} \bigcirc \mathbf{b}$ is available, the resulting JMED is given by $e(\mathbf{c}) = \epsilon_{\bigcirc}(e(\mathbf{a}), e(\mathbf{b}), \mathbf{c})$.

5.4.2.6. Example. Let Definition 5.4.2.2 hold for $\bigcirc = +$ as well as indexing functions f and h. From Theorem 5.1.2.1, if $\mathbf{c} = \mathbf{a} + \mathbf{b}$ is defined on $\mathbf{R}^{\mathbf{X}}$, then $\delta\mathbf{c}(\mathbf{x}) \leq \delta\mathbf{a}(\mathbf{x}) + \delta\mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}$. Let r denote a function that extracts an element in $h^{-1}(p_2(\mathit{domain}(e)))$, $s_a \equiv \mathbf{a}(\mathbf{x}) \cdot r(\mathbf{a}(\mathbf{x}))$, and $s_b \equiv \mathbf{b}(\mathbf{x}) \cdot r(\mathbf{b}(\mathbf{x}))$. The resulting JMED $e(\mathbf{c})$ is given by

$$e(\mathbf{c})(f(\mathbf{c}(\mathbf{x})), h(s_a + s_b)) = e(\mathbf{a})(f(\mathbf{a}(\mathbf{x})), h(s_a)) + e(\mathbf{b})(f(\mathbf{b}(\mathbf{x})), h(s_b)). \tag{226}$$

It is assumed that h contains a renormalizing function, in order to maintain the error fraction within the range [0,1].

The error predictors defined in Sections 5.1 through 5.3 are similarly expressed in terms of the JMED. We next discuss information-theoretic issues associated with error propagation.

5.5. Information Theory and Error-Tolerant Computation.

The information-theoretic description of computational error is scantily reported in the image processing literature. We thus present a summary of salient theory, with examples. A brief digression on applications of information theory to cryptography concludes this section.

5.5.1. Models of Information Transmission.

We begin with Shannon's information-theoretic model of communication, which we modify to be isomorphic to the commutativity diagram of Figure 1.

5.5.1.1. Observation. Per Shannon's seminal paper [68], we have the following diagram:

Figure 6. Information-theoretic model of communication.

which can be modified by adding an encoder and decoder, to yield the following model:

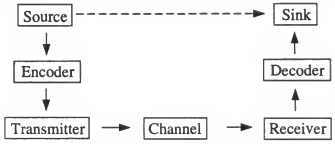


Figure 7. Information-theoretic model of communication.

where the dashed line denotes an idealized path.

5.5.1.2. Observation. An analogy can be drawn between the preceding diagram and the commutativity diagram of an isomorphic computational system, shown in Figure 1. For example, denote the source as an image $\mathbf{a}$; the sink $\hat{\mathbf{a}}$; the encoder and decoder as a transform T and its inverse T^{-1} (or an ϵ-near approximation to its inverse T^*), respectively; the encoder output $\mathbf{a}_c = T(\mathbf{a})$, and receiver input $\hat{\mathbf{a}}_c$. Letting an image operation $\bigcirc$ represent the idealized channel in Figure 7, we see that $\bigcirc'$, the analogue of $\bigcirc$ over *range*(T), can represent the corresponding real communication channel, which may include effects due to noise, system errors, etc.

5.5.1.3. Remark. The model described in the preceding observation is intuitively attractive for the analysis of both compressive and encryptive computation, and provides insights into certain aspects of cryptanalysis. In order to gain a unified understanding of such problems, we briefly address several related mathematical issues, as follows.

5.5.2. Basic Information Theory.

We begin with several observations from information theory, which prepare us for an information-theoretic description of image compression.

5.5.2.1. Observation. Assume that processes P_1 and P_2 have n_1 and n_2 equiprobable outcomes, respectively. If the outputs of P_1 and P_2 are combined to yield the output of a process P_3, then P_3 has $n_3 = n_1 \cdot n_2$ equiprobable outcomes. The computation of n_3 is described as follows.

Let $P_1, P_2, P_3 : S \rightarrow S$, and let $n_1 = |range(P_1)|$ with $n_2 = |range(P_2)|$. If P_3 is obtained from some function of the output of P_1 and P_2, then $domain(P_3) \subseteq range(P_1) \times range(P_2)$. However, we defined $range(P_i) = domain(P_i)$, for i = 1..3. Thus, the preceding expression can be rewritten as

$$|range(P_3)| \subseteq |range(P_1)| \times |range(P_2)|. \tag{227}$$

Assuming the limiting case, where the preceding relation is the equality relation, we have that $n_3 = n_1 \cdot n_2$.

5.5.2.2. Observation. The amount of information output by a continually productive source is a strictly increasing function of time.

5.5.2.3. Definition. Let H denote the information content (or *entropy*) of a process, and let H denote the operation that determines H. If $H_i = H(P_i)$, for i = 1..3, then $H_1 = H_2 + H_3$.

5.5.2.4. Assumption. Let the number of equiprobable outcomes of a process $P : S \rightarrow S$ be determined by a function $N : (S \rightarrow S) \rightarrow \mathbf{N}$. Assume that a function f combines two processes of form $S \rightarrow S$ to yield a process of form $S \rightarrow S$.

What analytically tractable function can embody the concept of N?

5.5.2.5. Lemma. Under constraint of Assumption 5.5.2.4, the entropy $H(P_3)$ of a process $P_3 = f(P_1,P_2)$ is given by $H(P_3) = \log(n_3) = \log(N(P_3))$ if the following conditions are satisfied:

(i) P_1,P_2: $S \rightarrow S$ are two processes having equiprobable outcomes denoted by $n_i = N(P_i)$, for i = 1..2.

(ii) P_3 has $n_3 = n_1 \cdot n_2$ equiprobable outcomes.

(iii) If $H_i = H(P_i)$, for i = 1..3, then $H_3 = H_1 + H_2$.

Proof. Assume that the givens and conditions i) through iii) hold. Note that $H = \log$ is the only function that satisfies i)-iii) simultaneously, since $n_3 = n_1 \cdot n_2$, $H(P_3) = \log(n_3) = \log(N(P_3))$, and

$$\log(n_3) = \log(n_1 \cdot n_2) = \log(n_1) + \log(n_2) . \qquad (228)$$

Thus, if $H = \log$, then $H_3 = H_1 + H_2$, and the lemma holds. □

5.5.2.6. Remark. Suppose we have two processes P_1 and P_2 with numbers of equiprobable outcomes n_1 and n_2. Now, let probabilities p_1 and p_2 be associated with n_1 and n_2, such that $n = n_1 + n_2$ and

$$p_1 = \frac{n_1}{n} \quad \text{and} \quad p_2 = \frac{n_2}{n}. \tag{229}$$

Recall that the entropy associated with one message among n equally likely outcomes is denoted by $H(n) = \log(n)$. However, for n_1/n of the time, $H(n_1) = \log(n_1)$. Symmetrically, $H(n_2) = \log(n_2)$. Thus, the net entropy is given by

$$\begin{aligned} H &= \log(n) - (n_1/n) \cdot \log(n_1) - (n_2/n) \cdot \log(n_2) \\ &= -p_1 \cdot \log(p_1) - p_1 \cdot \log(p_1). \end{aligned} \tag{230}$$

Since $p_1, p_2 < 1$, the logarithms are negative, and $H > 0$.

Via Equation 230, we can describe information content in terms of a probability distribution $\{p_1 \in [0, 1] : i = 1..n\}$, as shown in the following theorem.

5.5.2.7. Theorem. Let a process P have n possible outcomes that exhibit individual probabilities p_i, where $i = 1..n$. The entropy of P, denoted by $H(P)$, is given by:

$$H(P) = \sum_{i=1}^{n} -p_i \cdot \log(p_i). \tag{231}$$

Proof. The proof, which is well known, follows from Remark 5.5.2.6 and induction on □

We next restate the law of additive combination of information, with proof given in References 30, 69, and 70.

5.5.2.8. Theorem. If a process P consists of two statistically independent processes P_1 and P_2, then the entropy of P is given by

$$H(P) = H(P_1) + H(P_2). \tag{232}$$

5.5.2.9. Theorem. If a process P consists of two processes P_1 and P_2 that are not statistically independent, then the entropy of P is given by

$$H(P) < H(P_1)+H(P_2). \tag{233}$$

The following theorem of maximum information is stated without proof, which is given in Reference 69.

5.5.2.10. Theorem. If a discrete process P is a source that has n probable outcomes that exhibit individual probabilities p_i, where i = 1..n, then P yields maximum information when $p_i = 1/n$, for all $1 \leq i \leq n$.

The following examples suffice to show the utility of the maximum-information theorem.

5.5.2.11. Example. The modern English alphabet, denoted by $\mathbf{F}$, has 26 symbols. Thus, $H(\mathbf{F}) = \log(26) \approx 4.5$ bits/symbol. If the probability of each letter is taken into account and we apply Equation 233, then $H(\mathbf{F}) < 4$ bits/symbol. The actual information content of English has been shown to be approximately one bit per symbol [30], since the nonuniform occurrence of n-grams implies statistical dependence among symbols.

5.5.2.12. Example. Assume that the substitution cipher $T_{1s}: \mathbf{F} \times \mathbf{K}^{|\mathbf{F}|} \rightarrow \mathbf{F}$ is one-to-one and onto. Let $|\mathbf{F}| = 26$, as is customary, and let the keyspace $\mathbf{K} = \mathbf{F}$. Thus, there are 26! (i.e., $|\mathbf{F}|!$) possible keys of length 26 symbols.

Let us construct a 26-character key that encrypts a 40-character text. The information content of the key is given by $H_k = \log(|\mathbf{F}|!) = \log(26!)$. If the text has entropy H_t, then the substitution has entropy $H(T_{1s}) < H_t + H_k$.

From Theorem 5.5.2.7, a 40-character text has maximum entropy $\forall(H_t) = 40 \cdot \log(26)$. Since we cannot determine a priori whether the plaintext and key are statistically independent, we use Theorems 5.5.2.7 and 5.5.2.9 to derive the inequality

$$\forall(H(T_{1s})) \geq H_t + H_k = H_t + \log(26!). \tag{234}$$

Substituting the maximum entropy for H_t, we obtain:

$$H_t \leq V(H_t) - H_k = 40 \cdot \log(26) - \log(26!) \approx 100 \text{ bits.} \tag{235}$$

At 100 bits per 40 characters, we have 2.5 bits/symbol, which is considerably less than the maximum information content of text computed in Example 5.5.2.11.

5.5.3. Information Theory and Data Compression

The preceding examples suggest that one could construct transformations that compress the information content of a given source. We next show that a compressive transform can exploit existing communication channel bandwidth to increase a channel's information-carrying ability.

5.5.3.1. Definition. An *ideal channel* transmits a finite number of symbols without error, given the required transmission time of t_i seconds per symbol s_i.

5.5.3.2. Definition. *Channel capacity* is the maximum rate (in bits/second) at which one can transit a signal over a channel using the fastest possible source.

5.5.3.3. Remark. Channel capacity may be effectively increased by placing an encoder between the source and the channel. We initially assume that the encoder is lossless (i.e., is an invertible transformation). In the modification of Figure 7 discussed in Section 5.5.1, such an encoder would correspond to the transformation T.

5.5.3.4. Definition. A *statistical encoder* outputs symbols $s_i \in \mathbf{F}$, where each symbol has probability $p_i = \Pr(s_i)$. If the output $\mathbf{a}_c$ of a statistical encoder consists of n symbols, then $\mathbf{a}_c$ is called an *n-gram*.

5.5.3.5. Definition. The *compression ratio of a statistical encoder* that outputs an n-gram is computed as:

$$CR = \left[\frac{1}{n} \cdot \sum_{i=1}^{n} p_i \cdot siz(s_i) \right]^{-1}, \tag{236}$$

where *siz* was defined in Chapter 3.

5.5.3.6. Example. Assume that a source has an alphabet $\mathbf{F} = \{a,b\}$, where the probabilities $Pr(a) = 0.7$ and $Pr(b) = 0.3$ are assumed to be statistically independent, for purposes of convenience. The source output rate is assumed to be 80 symbols per minute.

The entropy of this source is given by:

$$\begin{aligned} H &= -0.7 \cdot \log(0.7) - 0.3 \cdot \log(0.3) \\ &= 0.779 \text{ bits/symbol}, \end{aligned} \tag{237}$$

and the information rate is given by:

$$\frac{H}{t} = \frac{0.779 \text{ bits}}{\text{symbol}} \cdot \frac{80 \text{ symbols/min}}{60 \text{ seconds/min}} = 1.038 \text{ bits/sec}. \tag{238}$$

However, we prefer the channel to transmit $\mathbf{F}$-valued messages at 0.95 bits/sec, which is less than the source output of 1.038 bits/sec. Thus, we derive an encoder that constrains the source to exhibit an output bandwidth that is less than or equal to the channel bandwidth.

Since $Pr(a) \neq Pr(b)$, we can employ a digraphic encoder $T_{dg}: \mathbf{F}^2 \rightarrow \{0,1\}^2$ to describe data via unique prefix codes. For example, T_{dg} could implement the following map:

aa $\mapsto$ 0	with probability	$0.49 = 0.7^2$
ab $\mapsto$ 10	" "	$0.21 = 0.7 \cdot 0.3$
ba $\mapsto$ 110	" "	$0.21 = 0.3 \cdot 0.7$
bb $\mapsto$ 111	" "	$0.09 = 0.3^2$

From Definition 5.5.3.5, the inverse compression ratio of T_{dg} is given by

$$\begin{aligned} CR^{-1} &= \frac{1}{m} \cdot \sum_{i=1}^{m} p_i \cdot d_i \\ &= 0.5 \cdot (0.49 \cdot 1 + 0.21 \cdot 2 + 0.21 \cdot 3 + 0.09 \cdot 3) \\ &= 0.5 \cdot 1.81 = 0.905 \text{ encoding symbols/source symbol}, \end{aligned} \tag{239}$$

which implies CR = 1.105. The source-encoder combination will now produce the output rate (H/t)/CR = 1.038 · 0.905 = 0.939 bits/sec. Thus, the channel, with a capacity of only 0.95 bits per second, can transmit the output of the combined source and encoder, which has a rate less than 0.94 bits per second.

We next discuss issues related to the transmission of encoded signals sent across realistic channels.

5.5.4. Information Loss and Computational Error.

In practice, it is useful to express the effect of noise on encoded signals in terms of information-theoretic measures such as entropy. Such measures depend upon noise statistics and correlation of noise sources with encoded data. Additionally, information-theoretic analyses that describe channel errors can describe computational error.

5.5.4.1. Observation. Let a message or image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ be specified as $\mathbf{a} = (s_1, s_2, \ldots, s_n)$. In the preceding section, we compressed a signal using a statistical encoder to send N bits per second of information across a channel with bandwidth M < N. In realistic channels, noise often corrupts K < M bits of **a**. For example, the noise may be correlated with **a** or the encoded image $\mathbf{a}_c$. Additionally, the noise may be nonuniformly distributed in time or space.

5.5.4.2. Remark. When such noise is modelled as being uniformly distributed (for purposes of convenience) and is random (and thus is irreducible), we can say that the entropy $H(\mathbf{a})$ is biased or increased in the communication channel by K bits per symbol. Using the formalism derived in Section 5.5.1, if a signal **a** is compressed to yield $\mathbf{a}_c$ at the channel input, then the entropy at the receiver input is given by

$$\begin{aligned} H(\hat{\mathbf{a}_c}) &= H(\mathbf{a}_c) + \mathrm{K} \\ &= H(\mathbf{a})/\mathrm{CR} + \mathrm{K}, \end{aligned} \tag{240}$$

where CR denotes the compression ratio. If the symbol probabilities are not independent, then we replace equality with "<", per Theorem 5.5.2.9.

When the source rate N is less than the channel bandwidth M but the source and noise rates together (N + K) < M and K < N, portions of the source will be replaced by noise. As a result, we have the following theorem.

5.5.4.3. Theorem. Let a signal **a** be transmitted along a lossy channel of bandwidth B > $H(\mathbf{a})$ = N. If the channel has K < B irreducible noise rate and K < N, then the following statements hold:

(i) L = B − N bits per symbol represent noise that does not necessarily corrupt **a**;

(ii) B − K bits of **a** are not necessarily corrupted by noise; and

(iii) K − L bits of **a** represent noise.

Proof. Let the givens hold and be portrayed by Figure 8, from which i) through iii) follow directly. □

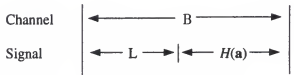


Figure 8. Arrangement of noise levels in a signal **a** with entropy $H(\mathbf{a})$ that partially occupies a channel of bandwidth B having K noise bits.

5.5.4.4. Example. In Example 5.5.3.6, an encoder compressed a 1.04 bits/pixel (bpp) signal **a** into a 0.94 bpp signal $\mathbf{a}_c$, which could then be transmitted along a 0.955 bpp channel. If the channel has noise of K = 0.1 bpp, then $H(\mathbf{a}_c)$ is increased by K if the noise is additive, or 0.1 bpp of $\mathbf{a}_c$ are replaced by noise (i.e., become irreversibly noise-corrupted).

5.5.4.5. Observation. Theorem 5.5.4.3 says nothing about the spatial effects of noise on compressed imagery. For example, assume an image $\mathbf{a}$ is transmitted along a noisy channel in uncompressed form. Let errors in the source image are highly correlated with bit errors in the compressed image $\mathbf{a}_c$. Thus, the distortions in $\mathbf{a}_c$ will represent the effect that such distortions would have on $\mathbf{a}$. However, the majority of commonly-used compressed schemes implement decorrelate $\mathbf{a}$ and $\mathbf{a}_c$ in order to achieve high compression. Unfortunately, if noise statistics and ergodicity are not well known a priori, unpredictable (and possibly serious) image distortions can result. The following example is illustrative.

5.5.4.6. Example. Consider that eight contiguous bits of a VPIC-compressed image could be occupied by (a) one flag bit indicating the presence of an edge or mean block, (b) four bits describing the block mean, and (c) three bits describing the block gradient intensity *if the block is an edge block.* Otherwise, the last three bits would commence with the flag bit of the next block. Since mean and edge blocks have different lengths, an error in the flag bit could erroneously perturb the remainder of the decoding process. As a result, we recommend complete error protection for the flag bit, as well as high error protection for the more significant bits of the mean, gradient intensity and orientation, and full error protection for the VPIC exemplar index. We further discuss such requirements in Chapters 9 and 10.

The preceding formalisms can be used to describe computational error in terms of SNR.

5.5.4.7. Observation. If we replace the realistic channel of Figure 7 with an operation $\bigcirc'$ that computes over the range space of an encoding transform T: S $\rightarrow$ S′, and $\bigcirc'$ is the analogue of an operation $\bigcirc$ over *domain*(T), then we can model error inherent in a realistic computational system $X = (\{S, \bigcirc\}, \{T, \Theta\}, \{S', \bigcirc'\}, T^*)$, per the concepts developed earlier in this chapter. In such cases, Theorem 5.5.4.3 describes information loss in X, but does not describe the spatial phenomena associated with such error.

Since it is customary to discuss error in terms of SNR, we present the following definition.

5.5.4.8. Definition. Let a real-valued source image with m-bit pixels be corrupted by $k < m$ bits of noise per pixel. If such error ϵ occurs in the less significant bits and is uniformly distributed, then the associated SNR is given by

$$SNR_\epsilon = \frac{2^m}{2^k} = 2^{m-k}. \tag{241}$$

This concludes our discussion of information loss.

CHAPTER 6
CLASS 1 TRANSFORMATIONS

This chapter contains derivations of analogues to the image algebra pointwise, global reduce, and image-template operations over the range spaces of substitutional ciphers (Section 6.1), transpositional ciphers (Section 6.2), and linear transforms (Section 6.3).

6.1. Substitutional Ciphers.

The substitutional cipher $T_{1s}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is a bijection and, therefore, an isomorphism. As a result, operations over *range*(T_{1s}) are arrived at merely by relabelling the operands of T_{1s}. We first present high-level theory, then examine the derivation of transform-regime analogues and duals over *range*(T_{1s}).

6.1.1. Basic Theory.

Since T_{1s} is a bijection, the derivation of operations over the transform's range space is relatively simple, and furnishes a useful introduction to the organization of our derivational techniques.

6.1.1.1. Definition. The substitutional cipher $T_{1s}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is a bijection that is based upon the scalar grey-level mapping

$$g: \mathbf{F} \rightarrow \mathbf{F} \tag{242}$$

and can be decomposed as

$$^{\mathrm{M}}T_{1s}: \mathbf{F} \rightarrow \mathbf{F} \;\triangleq\; {}^{\mathrm{F}}T_{1s}: \mathrm{f} \mapsto g(\mathrm{f}),\ \mathrm{f}, \mathrm{g} \in \mathbf{F}. \tag{243}$$

Since T_{1s} is a bijection, it is an isomorphism with inverse $T_{1s}^{-1}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$, which is implemented at fine granularity in terms of $g^{-1}: \mathbf{F} \rightarrow \mathbf{F}$.

6.1.1.2. Assumption. Let $\mathbf{a}, \mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ denote two source images that are transformed by T_{1s} to yield the corresponding images $\mathbf{a}_c, \mathbf{b}_c \in \mathbf{F}^{\mathbf{X}}$, and let the pointwise image operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$.

6.1.1.3. Lemma. Under constraint of Definition 6.1.1.1 and Assumption 6.1.1.2, a dual $\bigcirc': \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ of $\bigcirc$ over *range*(T_{1s}) is given by:

$$\mathbf{c} = \bigcirc'(\mathbf{a}_c, \mathbf{b}_c) = \bigcirc\left(g^{-1}(\mathbf{a}_c), g^{-1}(\mathbf{b}_c)\right). \tag{244}$$

In other words, $\bigcirc' = \bigcirc \circ g^{-1}$.

Proof. The proof follows directly from the fact that T_{1s} has a bijection g as its basis function, and T_{1s} is thus a bijection. □

6.1.1.4. Example. Let $\mathbf{a}$ = (1,2,0) and let $T_{1s}(x) = x+1 \ mod\ 3$. Thus, $\mathbf{a}_c = T_{1s}(\mathbf{a})$ = (2,0,1). Similarly, if $\mathbf{b}$ = (2,1,0), then $\mathbf{b}_c$ = (0,2,1). Let $\bigcirc = +_3$ (i.e., addition modulo 3) such that $\bigcirc(\mathbf{a},\mathbf{b}) = \mathbf{a} +_3 \mathbf{b}$ = (0,0,0). We can express $\bigcirc$ in tabular form as

$\bigcirc$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Similarly, T_{1s} has as its basis function g, which has an inverse g^{-1}. Both operations are depicted in tabular form as

g	0	1	2
X	1	2	0

g^{-1}	0	1	2
X	2	1	0

Applying g^{-1} to $\bigcirc$ (per Equation 244), we express the tabular representation of $\bigcirc'$ as

$\bigcirc'$	0	1	2
0	1	0	2
1	0	2	1
2	2	1	0

which is obtained by rotating the tabular representation of $\bigcirc$ about its minor diagonal.

6.1.1.5. Lemma. Under constraint of Definition 6.1.1.1 and Assumption 6.1.1.2, the analogue $\bigcirc'\colon \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ of $\bigcirc$ over *range*(T_{1s}) is given by:

$$\mathbf{c} = T_{1s}^{-1}(\bigcirc'(\mathbf{a}_c, \mathbf{b}_c)) = g(\bigcirc(g^{-1}(\mathbf{a}_c), g^{-1}(\mathbf{b}_c))). \tag{245}$$

That is, $\bigcirc' = g \circ \bigcirc \circ g^{-1}$.

Proof. The proof follows directly from the proof of Lemma 6.1.1.3. □

We next describe techniques for deriving pointwise duals and analogues over *range*(T_{1s}).

6.1.2. Technique: Binary Pointwise Operations.

Assuming that $T_{1s}\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ is a finite mapping, the technique for deriving duals (and, therefore, analogues) over *range*(T_{1s}) proceeds as follows.

Step 1. Assume that $\bigcirc$, a binary pointwise operation over *range*(T_{1s}), is specified in terms of its matrix representation **d**, as discussed in Section 4.4. Additionally, we assume that g is the basis function of T_{1s}.

Step 2. Via an indexing bijection $h\colon \mathbf{F} \to \Lambda$, where $\Lambda = [1, |\mathbf{F}|]$ we set *range*(*choice*(*domain*(T_{1s}))) = *range*(*choice*(*range*(T_{1s}))) = Λ. We similarly transform $g\colon \mathbf{F} \to \mathbf{F}$ into $f\colon \Lambda \to \Lambda$.

Step 3. Applying f to each projection of the coordinates of matrix representation **d**, we obtain the matrix representation of $\bigcirc'$, the dual of $\bigcirc$.

Thus, we use the indexing function h to map a greylevel transformation (normally applied to an image) into a spatial transform of the matrix representation of $\bigcirc$.

6.1.2.1. Lemma. Under constraint of Definition 6.1.1.1 and Assumption 6.1.1.2, let $h: \mathsf{F} \to \Lambda$, where $\Lambda = [1, |\mathsf{F}|] \subset \mathbb{N}$, transform the basis function $g: \mathsf{F} \to \mathsf{F}$ of T_{1s} into $f: \Lambda \to \Lambda$. If $\mathbf{d} \in \Lambda^{\Lambda \times \Lambda}$ is the matrix representation of $\bigcirc$, then $\mathbf{g}$, the matrix representation of $\bigcirc'$ (which denotes the dual of $\bigcirc$ over $range(T_{1s})$), is given by:

$$\mathbf{g} = \mathbf{d} \circ f^{-1}. \tag{246}$$

It follows that

$$\bigcirc'(T_{1s}(\mathrm{f}), T_{1s}(\mathrm{g})) = h^{-1}(\mathbf{g}(h(\mathrm{f}), h(\mathrm{g}))), \forall \mathrm{f}, \mathrm{g} \in \mathsf{F}. \tag{247}$$

Proof. Assuming that the givens hold, f and h have inverses because they are bijections. If $\mathbf{g}$ is the matrix representation of $\bigcirc$, then from the definition of f,

$$\mathbf{g} = \mathbf{d} \circ f^{-1} = \{(\mathbf{w}, \mathbf{d}[f^{-1}(\mathbf{w})]) : \mathbf{w} \in \Lambda \times \Lambda\}. \tag{248}$$

The definition of $\bigcirc'$ follows directly from $\mathbf{g}$ and h. □

6.1.2.2. Theorem. Let Lemma 6.1.2.1 hold. If $\mathbf{g} \in \Lambda^{\Lambda \times \Lambda}$ denotes the matrix representation of the dual of $\bigcirc$, then $\bigcirc'$, the analogue of $\bigcirc$ over $range(T_{1s})$, is given by:

$$\bigcirc'(T_{1s}(\mathrm{f}), T_{1s}(\mathrm{g})) = g(h^{-1}[\mathbf{g}(h(\mathrm{f}), h(\mathrm{g}))]), \forall \mathrm{f}, \mathrm{g} \in \mathsf{F}. \tag{249}$$

Proof. The proof follows directly from Lemma 6.1.2.1, the definitions of transform-regime dual and analogue operations, and the definition of h and g. □

6.1.2.3. Complexity. In the preceding lemmas, $\mathbf{d}$ can be obtained computationally in $|\mathsf{F}^2|$ computations of $\bigcirc$. Application of the spatial transform f^{-1} yields the dual $\bigcirc'$ in $|\mathsf{F}^2|$ I/O operations. The analogue is obtained from the dual by $|\mathsf{F}^2|$ applications of h^{-1} and g. If $\mathbf{d}$ is stored in memory, then a one-time computation of $\bigcirc'$ is required.

6.1.3. Technique: Global Reduce Operations.

Given the preceding lemma and theorem, the global reduce operations are simply derived.

6.1.3.1. Theorem. If $\Gamma: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}$ is based upon the associative, commutative function $\gamma: \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$ and Theorem 6.1.2.2 holds, then $\Gamma': \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}$, an analogue of Γ over *range*(T_{1s})), is given by one of the following statements:

(i) $\Gamma' = g \circ \Gamma$; or

(ii) If $\gamma': \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$ denotes the analogue of γ over *range*(T_{1s})), then γ' is the basis function of Γ'.

Proof. Assuming that the givens hold, (i) follows directly from the definitions of Γ and g as well as the commutativity diagram of Figure 1. Statement (ii) follows directly from the definition of an analogue, as well as from the definition of a global reduce operation. □

6.1.3.2. Example. Let $T(x) = x+2$ *mod* k, where $x, \mathrm{k} \in \mathbf{Z}^+$ and $\mathrm{k} > 2$. Let $\mathbf{a} \in \mathbf{Z}_{\mathrm{k}}^{\mathbf{X}}$ be transformed to yield $\mathbf{a}_c = T(\mathbf{a})$, and let $\gamma = +_{\mathrm{k}}$ denote addition modulo k. For example, if $\mathbf{a}$ = (0,1,2,3,2,3,1,1,0) and k = 4, then $\mathbf{a}_c$ = (2,3,0,1,0,1,3,3,1). Taking the image sum yields $\Sigma\mathbf{a}$ = 13 and $\Sigma\mathbf{a}_c$ = 14. However, if γ is the basis of Γ, then $\Gamma(\mathbf{a}) = 0$. In order to obtain γ', the analogue of γ over *range*(T), we apply the technique of Section 6.1.1.3 to yield the following tabular representations:

γ	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

γ'	2	3	0	1
2	0	1	2	3
3	1	2	3	0
0	2	3	0	1
1	3	0	1	2

which means that $\gamma' = \gamma$, since the preceding matrices are block-symmetric. Thus,

$$\Gamma'(\mathbf{a}_c) = 2\,\gamma'\,3\,\gamma'\,0\,\gamma'\cdots\gamma'\,1 = 2\,. \tag{250}$$

As a result, we have $\Gamma'(\mathbf{a}_c) = T(\Gamma(\mathbf{a}))$, which verifies that γ' is the analogue of γ over *range*(T).

However, as we shall see in Chapter 7, if $\gamma = +_m$, where $m \neq k$, then γ' may not exist, due to the induction of a many-to-one mapping in $T(\Gamma(\mathbf{a}))$. Rather than deducing the analogue from the dual, we provide the reader with the converse method.

6.1.3.3. Theorem. Let the preceding lemmas hold, and let Ω denote an analogue of Γ over *range*(T_{ls}). Then, $\Gamma'\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}$, a dual of Γ over *range*(T_{ls}), is given by $\Gamma' = g^{-1} \circ \Gamma$.

Proof. Assuming that the givens hold, the proof follows directly from the fact that g is a bijection. □

6.1.3.4. Complexity. The associative, commutative function γ' can be obtained in $|\mathbf{F}|^2$ computations of g and γ. If γ is stored, then this is a one-time computation. The preceding example is illustrative.

6.1.4. Technique: Image-Template Operations

Given the dual (analogous) operations of pointwise arithmetic and global reduce functions, the construction of various analogues of the image-template operations over *range*(T_{1s}) is a trivial matter of functional composition.

6.1.4.1. Assumption. Under constraint of Definition 6.1.1.1 and Assumption 6.1.1.2, let the operation $\bigcirc : \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ combine a source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and a template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ to yield the image $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{t}) = \mathbf{a} \textcircled{\scriptsize T} \mathbf{t}$. Assume that $\textcircled{\scriptsize T}$ consists of two associative, commutative operations $\circ, \gamma : \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$, the latter of which is the basis function for the global reduce function $\mathbf{\Gamma}$. Let $\circ' = g(\circ)$ and $\gamma' = g(\gamma)$ denote the duals (analogues) of $\circ$ and γ' over *range*(T_{1s}).

6.1.4.2. Theorem. Under constraint of Assumption 6.1.4.1, if $\bigcirc' : \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$, the analogue of $\bigcirc$ over *range*(T_{1s}), combines a ciphertext image $\mathbf{a_c} = T_{1s}(\mathbf{a})$ and a template $\mathbf{s} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ to yield the image $\hat{\mathbf{a}}_c = \bigcirc'(\mathbf{a}_c,\mathbf{s}) = T_{1s}(\mathbf{a} \textcircled{\scriptsize T} \mathbf{t})$, then $\bigcirc'$ is given by:

$$\begin{aligned} \hat{\mathbf{a}}_c &= \bigcirc'(\mathbf{a}_c,\mathbf{s}) = \mathbf{a}_c \textcircled{\scriptsize T}' \mathbf{s} \\ &\equiv \bigcup_{\mathbf{y} \in \mathbf{X}} \left(\mathbf{y}, \underset{\mathbf{x} \in \mathbf{X}}{\mathbf{\Gamma}'} \mathbf{a}_c(\mathbf{x}) \circ' \mathbf{s}_{\mathbf{y}} \right), \end{aligned} \tag{251}$$

where $\mathbf{s}$ is defined by its weights as $\mathbf{s}_{\mathbf{y}}(\mathbf{x}) = g(\mathbf{t}_{\mathbf{y}}(\mathbf{x}))$, $\forall \mathbf{x},\mathbf{y} \in \mathbf{X}$. More simply, $\mathbf{s} = g(\mathbf{t})$.

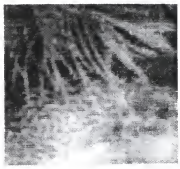
Proof. The proof follows directly from the discussion of image-template operations in Section 3.4. Note that the equivalence operation is employed, since Equation 251 forms the graph of $\mathbf{a}_c$. □

6.1.4.3. Complexity. Assume that the operations $\bigcirc'$ and γ' need not be computed for each invocation of Equation 251 (i.e., are stored in LUTs or are available on an encryptive processor). Then, it follows from our preceding discussion of complexity (Chapter 4) that the overhead associated with Equation 251 is bounded as $2{\cdot}|\mathbf{X}| \leq W_{\bigcirc'} \leq |\mathbf{X}| \cdot (|\mathbf{X}|+1)$ invocations of g associated with template conversion. Otherwise, the overhead increases to $3{\cdot}W_{\bigcirc'}$ (i.e., one instance of g per $\bigcirc'$, $\mathbf{\Gamma}'$, and the previously-given conversion of $\mathbf{t}$ to $\mathbf{s}$). Since the transform is not compressive, a sub-unitary speedup would be achieved by encryptive processing if $\bigcirc'$ and $\mathbf{\Gamma}'$ were not encoded in terms of one or more LUTs.

6.1.4.4. Example. Let $\mathbf{a} \in \mathbf{Z}_{256}^{\mathbf{X}}$, as shown in Figure 9a, be convolved with a von Neumann template $\mathbf{t}$ shown in Figure 9b. Let the transform $T(x) = (x+128) \ mod \ 256$ be applied to $\mathbf{a}$ to yield $\mathbf{a}_c \in \mathbf{Z}_{256}^{\mathbf{X}}$, as shown in Figure 9c. The transformed template $\mathbf{s}$, produced by the method of Equation 251, is shown in Figure 9d. Let the operation $\oplus' \colon \mathbf{Z}_{256}^{\mathbf{X}} \times (\mathbf{Z}_{256}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{Z}_{256}^{\mathbf{X}}$ be derived from analogues of addition and multiplication, as shown in Sections 6.1.2. Applying $T^{-1}(x) = (x{-}128) \bmod 256$ to the result $\mathbf{b}_c = \mathbf{a}_c \textcircled{\gamma}' \mathbf{s}$ (shown in Figure 9e) yields the image of Figure 9f, which portrays the image operation $\mathbf{b} = \mathbf{a} \oplus \mathbf{t}$.

6.1.5. Data Security and Information Loss.

The substitutional cipher is insecure due to high input-output correlation and preservation of the input histogram. Since T_{1s} is a finite bijection, no information loss occurs. Under constraint of Definition 6.1.1.1 and Assumption 6.1.1.2, we present the following cryptanalytic methods.



(a)

0	1	0
1	2	1
0	1	0

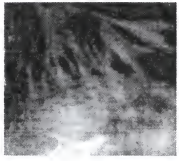
(b)

(c)

128	129	128
129	130	129
128	129	128

(d)

(e)



(f)

Figure 9. Local averaging of a noisy, two-dimensional, 8–bit image transformed by $T(x) = (x+128) \ mod \ 256$: (a-b) source image and template $\mathbf{a}$ and $\mathbf{t}$; (c-d) transformed image and template $\mathbf{a}_c$ and $\mathbf{s}$; (e) transformed result $\mathbf{b}_c$; (f) image-domain result $\mathbf{b} = T^{-1}(\mathbf{b}_c)$.

6.1.5.1. Observation. Assume that plaintext and ciphertext are available to a cryptanalyst. Since T_{1s} merely relabels the plaintext symbols in $\mathbf{a}$, one needs only to formulate an indexing function $h : \mathsf{F} \rightarrow \Lambda$, where $\Lambda = [1, |\mathsf{F}|] \subset \mathbb{N}$, as well as the trial plaintext $\mathbf{a} = \{(x, f) : x = h(f), f \in \mathsf{F}\}$. Application of $\mathbf{a}_c = T_{1s}(\mathbf{a})$ enables construction of the bijection $g : \mathbf{a}(\mathbf{x}) \vdash \mathbf{a}_c(\mathbf{x}), \forall \mathbf{x} \in \mathbf{X}$ which is implemented in ${}^{\mathsf{F}}T_{1s}$.

6.1.5.2. Observation. Assume the more difficult case, where the plaintext $\mathbf{a}$ is unknown, i.e., only the ciphertext $\mathbf{a}_c$ is available. Let $\mathbf{h}(\mathbf{a})$ and $\mathbf{h}(\mathbf{a}_c)$ denote the plaintext and ciphertext histograms. We use the following algorithm, which is a slight modification of a procedure presented in Sinkov [27]:

Step 1. For each $f \in \mathsf{F}$, compute the probabilities

$$\mathbf{b}(f) = \frac{\mathbf{h}(\mathbf{a}_c)(f)}{\Sigma \mathbf{h}(\mathbf{a}_c)}. \tag{252}$$

Step 2. Given the probability distribution $Pr(f)$, $f \in \mathsf{F}$, accumulate for each f the set

$$A_f = domain\left(\chi_{[Pr(f)-\epsilon, Pr(f)+\epsilon]}(\mathbf{b})\right), \tag{253}$$

which are the more probable candidates for substitution.

Step 3. Starting with the first letter f of the most frequent words (in colloquial English: *a* or *the*), mark the positions $D = domain(\mathbf{a}_c|_{\in A_f})$.

Step 4. Interrogate each unmarked successor value $g = \mathbf{a}_c(x+1)$ of each marked value $\mathbf{a}_c(x)$, where $x \in D$, to determine if $\mathbf{b}(g)$ matches (or nearly matches) the probabilities of any successors of f in a table of digram frequencies. If there is a match (near-match), then the decoded value $\mathbf{d}(x)$ is assigned the value f determined in Step 3.

Step 5. Repeat Steps 3 and 4 iteratively until the number of decoded words that correspond to known plaintext words (obtained from a lexicon) is maximized. As a result, $\mathbf{d}$ will contain a best-match candidate solution to $\mathbf{a}_c$.

This method can be implemented automatically for many natural languages. Convergence efficiency tends to increase when Pr(f) accurately portrays the plaintext statistics.

In the previous section, we introduced the reader to our derivation methodology. For purposes of brevity, we henceforth present more concise derivations, with the preceding methodology understood.

6.2. Transpositional Ciphers.

As discussed in the introduction, a transpositional cipher $T_{\mathrm{lt}}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is based upon a bijection $f: \mathbf{X} \rightarrow \mathbf{X}$. Such spatial bijections preserve image values and thus preserve both global reduce and pointwise operations. However, m-ary pointwise and image-template operations over T_{lt} require that the same transposition be applied to the domain of each input (plaintext) image.

6.2.1. Technique: Image-Template Operations.

Since pointwise and global reduce operations are not interesting cases of the pointwise transpositions, we provide a summary lemma, then briefly discuss image-template operations.

6.2.1.1. Definition. A *transpositional cipher* $T_{\mathrm{lt}}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ is based upon a spatial bijection $f: \mathbf{X} \rightarrow \mathbf{X}$, such that $\mathbf{a}_c = T_{\mathrm{lt}}(\mathbf{a}) = \mathbf{a} \circ f$.

6.2.1.2. Lemma. Under constraint of Definition 6.2.1.1, the following statements hold:

(i) A pointwise image operation $\bigcirc: (\mathbf{F}^{\mathbf{X}})^{\mathbf{m}} \rightarrow \mathbf{F}^{\mathbf{X}}$ exhibits a dual (analogous) operation over *range*(T_{lt}) that is given by $\bigcirc' = \bigcirc$. Here, we assume that the operands of pointwise binary operations are each transformed by the same instance of T_{lt}.

(ii) A global reduce operation $\mathbf{\Gamma}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}$ exhibits a dual (analogous) operation over *range*(T_{lt}) that is given by $\mathbf{\Gamma}' = \mathbf{\Gamma}$.

(iii) The matrix representations of dual and analogous operations of a given pointwise or global reduce operation are not equal.

Proof. Statements (i) and (ii) follow directly from our preceding discussion of lossy neighborhood operations (Section 3.4). Statement (iii) follows from the fact that input values are preserved in *range*(T_{lt}), but their order is perturbed by T_{lt}. □

Since the pointwise and global reduce operations are preserved over *range*(T_{lt}), one needs only to apply the spatial transform $f : \mathbf{X} \rightarrow \mathbf{X}$ to the template domain to produce a template that can be convolved over the ciphertext $\mathbf{a}_c = T_{\mathrm{lt}}(\mathbf{a})$.

6.2.1.3. Theorem. Let the operation $\bigcirc : \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ combine a source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and a template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ to yield the image $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{t}) = \mathbf{a} \textcircled{?} \mathbf{t}$. Under constraint of Definition 6.2.1.1 and the assumption that $\mathbf{a}_c = T_{\mathrm{lt}}(\mathbf{a})$, the operation $\bigcirc' : \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$, a dual (analogue) over *range*(T_{lt}) of $\bigcirc$, is given by:

$$\begin{aligned} \hat{\mathbf{a}}_c &= \bigcirc'(\mathbf{a}_c, \mathbf{t} \circ f) = \mathbf{a}_c \textcircled{?} (\mathbf{t} \circ f) \\ &\equiv \bigcup_{\mathbf{y} \in \mathbf{X}} \left(\mathbf{y}, \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{a}_c(\mathbf{x}) \cdot (\mathbf{t}_{\mathbf{y}} \circ f)(\mathbf{x}) \right), \end{aligned} \tag{254}$$

where $\mathbf{t} \circ f$ implies the existence of a weight matrix transformation $\mathbf{t}_{\mathbf{y}} \circ f$ for each $\mathbf{y}$ in $\mathbf{X}$.

Proof. Assuming that the givens hold, it is easily verified that the template $\mathbf{t}$, which is configured for the source domain $\mathbf{X}$, is not configured for the transformed domain $f(\mathbf{X})$. Since $\mathbf{t}$ may be space-variant, we need to transform the domain of each image $\mathbf{t}_{\mathbf{y}} \in \mathbf{F}^{\mathbf{X}}$ by composition with f. Note that the equivalence relation is used since Equation 254 describes construction of the graph of $\hat{\mathbf{a}}_c$. □

6.2.1.4. Complexity. Equation 254 requires overhead of $|\mathbf{X}|$ compositions of f with *domain*($\mathbf{t}_{\mathbf{y}}$) in addition to the overhead of $\textcircled{?}$. Given the discussion of complexity in Section 4.3, the resultant work is bounded as $2 \cdot |\mathbf{X}| \leq W_{\bigcirc'} \leq |\mathbf{X}| \cdot (|\mathbf{X}|+1)$ I/O operations.

6.2.2. Data Security and Information Loss.

Since image values are preserved in *range*(T_{lt}) there is no information loss. In the introduction, we briefly mentioned that the data security of spatial transformations could be compromised via histogram-based attack, which holds particularly for spatial bijections, since the probability distributions of plaintext and ciphertext image values are identical.

6.2.2.1. Observation. Cryptanalysis of transpositional ciphers is trivial, and includes the following two cases:

1. If T_{lt} is available to the adversary, then one merely formulates the image $\mathbf{a} \equiv \{(\mathbf{x},\mathbf{x}) \colon \mathbf{x} \in \mathbf{X}\}$, and performs the operation $\mathbf{a}_c = T_{\mathrm{lt}}(\mathbf{a})$. The graph of the spatial bijection f, upon which T_{lt} is based, is given by $G(f) = \{(\mathbf{a}_c(\mathbf{x}), \mathbf{a}(\mathbf{x})) \colon \mathbf{x} \in \mathbf{X}\}$.
2. If T_{lt} is not available to the adversary, then cryptanalysis is still trivial, since *range*(T_{lt}) contains all plaintext image values. Thus, histogram- or correlation-based attack can yield the plaintext [27,30].

6.2.2.2. Remark. Regarding correlation-based attack, since T_{lt} is a bijection, the location of a neighborhood in the source image $\mathbf{a}$ is highly correlated with the location of the corresponding transformed neighborhood in $\mathbf{a}_c$. Thus, one has merely to use different neighborhood domains to detect neighborhoods in $\mathbf{a}_c$ that correlate well with each other. This is especially easy in the case of frequently-occurring digrams and trigrams, such as the English determiners *an* and *the*. However, given the success of histogram-based attack, which is simpler computationally, correlation-based attacks are of pedagogic interest only.

6.2.2.3. Observation. The case of image transpositions differs significantly from message encryption via transposition. For example, levels of message complexity can be characterized as (0) symbol (1) word, (2) phrase or clause, (3) sentence, (4) topic, and (5) document. In contrast, the structure of imagery is not apparent beyond the pixel level, where maximum likelihood estimates can be formulated over von Neumann neighborhoods [71]. Similarly, the co-occurrence of symbols in messages can be described in terms of digram and trigram statistics, which are well known for a variety of natural languages. In contrast, the form of symbols and higher-level structures in imagery is not well understood. For example, at a low level are pixel values, whose co-occurrence has not been extensively studied [71]. At higher levels of abstraction, imagery can be treated in terms of multiresolution constructs (e.g., image pyramids) that are derived from the source image mathematically, using simple operations. The application of n-gram analyses to multiresolution imagery is a topic of current interest to us, and has not been extensively reported in the literature. In summary, the application of transpositions to imagery can yield more secure encryption, in principle, due to the current lack of knowledge about pixel value co-occurrence. Such concepts have been implemented in ciphers that use extensive transposition and permutation of bits (e.g., DES).

We next progress to the somewhat more interesting case of linear transformations, which are widely employed in signal and image processing and frequently form the basis for image compression via transform coding.

6.3. Linear Transforms.

Due to the linearity property, pointwise and global reduce operations over the range space of linear transforms can be readily derived. However, duals or analogues of image-template operations are less easily derived.

6.3.1. Basic Theory.

We briefly review the concept of a linear transform and its decomposition.

6.3.1.1. Definition. Given an image $\mathbf{a} \in \mathbb{C}^{\mathbf{X}}$, a *linear transformation* $T_{1l}: \mathbb{F}^{\mathbf{X}} \rightarrow \mathbb{F}^{\mathbf{X}}$ is a mapping that exhibits the following *linearity property*:

$$T_{1l}((k_1 \cdot \mathbf{a}) + k_2) = (k_1 \cdot T_{1l}(\mathbf{a})) + k_2 \, , \, k_1, k_2 \in \mathbb{C} \, . \tag{255}$$

6.3.1.2. Observation. For purposes of convenience, we classify linear transforms as *pointwise* or *neighborhood* transformations. A pointwise linear transform operates upon a single value in each of its input images (e.g., $T(\mathbf{a},\mathbf{b}) = \mathbf{a} + \mathbf{b}/k$). A neighborhood transform is a linear combination of values located in a subset of the domain(s) of one or more operand images. For example, linear inner product transforms are included in the neighborhood transforms. We further note that, due to their many-to-one basis maps, linear neighborhood transforms cannot be inverted as can their pointwise counterparts.

Additionally, if a linear transform preserves the sum of its input in the transform output, then the global reduce function of image sum is readily derived. Similarly, addition and multiplication by a constant are preserved over the range space of a linear transform. In contrast, the preservation of nonlinear pointwise (and global reduce) operations is not supported by the linearity property. We thus begin our discussion with the simple case of linear operations over linear pointwise transformations.

6.3.2. Technique: Pointwise Operations.

We consider linear and nonlinear pointwise operations, which do not have elaborate analogues over the output of pointwise linear transforms.

6.3.2.1. Lemma. If $\bigcirc'$ denotes a linear operation over the range space of a linear transform T, then $\bigcirc' \circ T$ is a linear operation.

Proof. The proof follows from the fact that the composition of linear operations yields a linear operation. □

6.3.2.2. Example. Let $\mathbf{a}, \mathbf{b} \in \mathbf{R}^{\mathbf{X}}$ and let T denote a linear transform. Then, the analogue of the pointwise binary operation $\bigcirc = +$ is given by $\bigcirc' = +$, which follows directly from the linearity property of T. A similar case holds for multiplication by a constant.

We next derive a dual of Hadamard multiplication over the range space of a pointwise linear transform.

6.3.2.3. Lemma. Given Lemma 6.3.2.1, let images $\mathbf{a}, \mathbf{a_c}, \mathbf{b}, \mathbf{b_c} \in \mathbf{R}^{\mathbf{X}}$, where $\mathbf{a_c} = T_{1l}(\mathbf{a}) = c_1 \cdot \mathbf{a} + d_1$ and $\mathbf{b_c} = T_{1l}(\mathbf{b}) = c_2 \cdot \mathbf{b} + d_2$. If the image operation $\bigcirc = *$ such that $\mathbf{c} = \mathbf{a} * \mathbf{b}$, then the dual of Hadamard multiplication is given by:

$$\mathbf{c} = \bigcirc'(\mathbf{a_c}, \mathbf{b_c}) = \frac{\mathbf{a_c} * \mathbf{b_c} - d_1 * \mathbf{b_c} - d_2 * \mathbf{a_c} + d_1 d_2}{c_1 \cdot c_2}. \tag{256}$$

Proof. From the givens, we have

$$\mathbf{a_c} * \mathbf{b_c} = c_1 c_2 * \mathbf{a} * \mathbf{b} + c_1 d_2 * \mathbf{a} + c_2 d_1 * \mathbf{b} + d_1 d_2, \tag{257}$$

which can be solved to yield

$$\mathbf{a} * \mathbf{b} = \frac{\mathbf{a_c} * \mathbf{b_c} - d_1 c_2 * \mathbf{b} - d_2 c_1 * \mathbf{a} - d_1 d_2}{c_1 \cdot c_2}. \tag{258}$$

The preceding equation can be rendered nonrecursive by noting that

$$\mathbf{a} = \frac{\mathbf{a_c} - d_1}{c_1} \quad \text{and} \quad \mathbf{b} = \frac{\mathbf{b_c} - d_2}{c_2}. \tag{259}$$

Substituting Equation 259 into Equation 258 yields Equation 256. □

6.3.2.4. Complexity. Equation 256 requires five additions and multiplications per pixel. If $\mathbf{a}, \mathbf{b} \in \mathbf{R}^{\mathbf{X}}$, then the total work required by $\bigcirc'$ equals $4 \cdot |\mathbf{X}|$ multiplications and $5 \cdot |\mathbf{X}|$ additions. Given $4 \cdot |\mathbf{X}|$ processors, $\bigcirc'$ can be computed on a pipelined parallel processor in the time required for two multiplications and two additions, as shown in Figure 10.

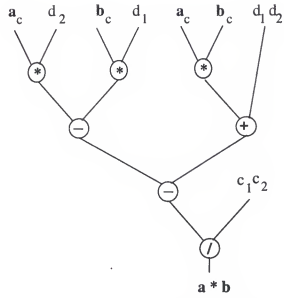


Figure 10. Dual of pointwise multiplication over the range space of the pointwise linear transform $T(x) = cx + d$.

6.3.2.5. Remark. The preceding lemma is not fanciful, but is useful for recovering the product of two images that have undergone linear contrast stretching. If one of the images is known (or approximately known) the other can be retrieved (approximated), as shown in Figure 11.

The analogue of pointwise multiplication over the range space of a linear pointwise transform $T(x) = cx + d$ follows from Lemma 6.3.2.3. The following lemma is illustrative.

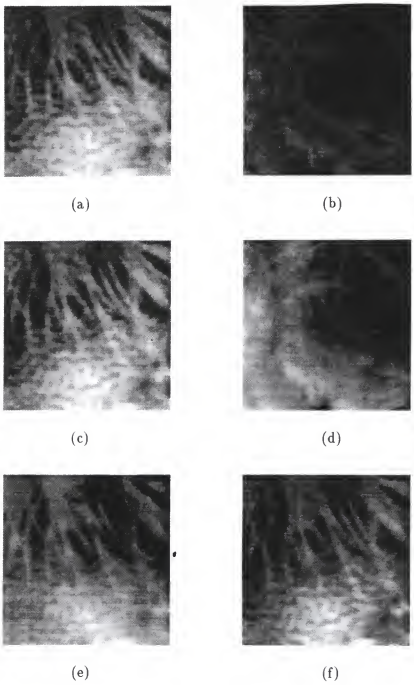


Figure 11. Recovery of source image **a** from image sum of contrast-stretched imagery: (a-b) source images **a** and **b**, (c-d) linear contrast-stretched images $\mathbf{a}_c = T(\mathbf{a})$ and $\mathbf{b}_c = T(\mathbf{b})$, (e) $\mathbf{c}_c = \mathbf{a}_c$ and $\mathbf{b}_c$, (f) recovery of **a** from $\mathbf{c}_c$ using $\mathbf{a} = T^{-1}(\mathbf{c}_c - T(\mathbf{b}))$.

6.3.2.6. Lemma. Given Lemma 6.3.2.3, let images $\mathbf{a}, \mathbf{a}_c, \mathbf{b}, \mathbf{b}_c \in \mathbf{R}^{\mathbf{X}}$, where $\mathbf{a}_c = T_{1i}(\mathbf{a}) = c \cdot \mathbf{a} + d$ and $\mathbf{b}_c = T_{1i}(\mathbf{b})$. If the image operation $\bigcirc = *$ such that $\mathbf{c} = \mathbf{a} * \mathbf{b}$, then the dual of Hadamard multiplication is given by:

$$\mathbf{c}_c = \bigcirc'(\mathbf{a}_c, \mathbf{b}_c) = \frac{\mathbf{a}_c * \mathbf{b}_c - d(\mathbf{a}_c + \mathbf{b}_c - d)}{c^2}. \tag{260}$$

Proof. The proof follows directly from the proof of Lemma 6.3.2.3 by substituting c for c_1 and c_2, as well as substituting d for d_1 and d_2. □

This simple result can be expanded to derive an analog of pointwise multiplication over the range spaces of all linear transforms, via the following decomposition.

6.3.2.7. Definition. Let the step function σ be given by

$$\sigma(\mathbf{y})(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{x} \geq \mathbf{y} \\ 0 \text{ otherwise} \end{cases}, \mathbf{x}, \mathbf{y} \in \mathbf{X}. \tag{261}$$

Alternatively, we can write σ as $\sigma(\mathbf{y}) = \left(\mathbf{1}|_{\{\mathbf{x}:\mathbf{x}\geq\mathbf{y}, \mathbf{y}\in\mathbf{X}\}}\right)\Big|^{\mathbf{0}}$.

6.3.2.8. Lemma. Given the step function σ, the *additive step function decomposition* of a one-dimensional image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ is given by

$$\mathbf{a} = \sum_{\mathbf{x}\in\mathbf{X}} (\mathbf{a} \oplus \mathbf{t})(\mathbf{x}) \cdot \sigma(\mathbf{x}), \tag{262}$$

where $\mathbf{t} = (-1\ \underline{1})$, with the target pixel italicized.

Proof. The proof sketch is embodied without loss of generality in the following example. Given the definition of σ and letting $\mathbf{a} = (c, d, e)$, we note that $\mathbf{b} = \mathbf{a} \oplus \mathbf{t} = (c, d-c, e-d)$. Thus, $\mathbf{a} = (c,c,c) + (0, d\text{-}c, d\text{-}c) + (0, 0, e\text{-}d)$

$$\begin{aligned} \mathbf{a} &= (c, c, c) + (0, d-c, d-c) + (0, 0, e-d) \\ &= (c,\ c+d-c,\ c+d-c+e-d) \\ &= (c, d, e), \end{aligned} \tag{263}$$

and the lemma holds. □

The preceding lemma is not limited to one dimensional images. For example, σ could be a step function with the same dimensionality as $\mathbf{X}$. However, one must define the maximum operation over $\mathbf{X}$ prior to implementing σ, which definition depends upon scanning order. For purposes of simplicity, we continue with the one-dimensional case.

6.3.2.9. Assumption. Let a linear transform $T : \mathbf{R}^{\mathbf{X}} \rightarrow \mathbf{R}^{\mathbf{X}}$ be applied to an image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ as $\mathbf{a}_c = \mathbf{a} \oplus \mathbf{r}$, where $\mathbf{r} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$. Assume that the inverse transformation $T^{-1} : \mathbf{R}^{\mathbf{X}} \rightarrow \mathbf{R}^{\mathbf{X}}$ is given by $\mathbf{a} = \mathbf{a}_c \oplus \mathbf{r}^{-1}$, such that $\mathbf{a} = \mathbf{a} \oplus \mathbf{r} \oplus \mathbf{r}^{-1}$.

6.3.2.10. Theorem. Under constraint of Definition 6.3.2.7, Assumption 6.3.2.9, and Lemma 6.3.2.8, let images $\mathbf{a}, \mathbf{a}_c, \mathbf{b}, \mathbf{b}_c \in \mathbf{R}^{\mathbf{X}}$, where $\mathbf{a}_c$ and $\mathbf{b}_c$ result from application of a linear transform T to $\mathbf{a}$ and $\mathbf{b}$. Let T be implemented as $T(\mathbf{a}) = \mathbf{a} \oplus \mathbf{r}$, where $\mathbf{r} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$, and let $\mathbf{s} = \mathbf{r}^{-1} \oplus \mathbf{t}$, where $\mathbf{t} = (-1\ \mathit{1})$, with the target pixel italicized. An analogue $\bigcirc' : \mathbf{R}^{\mathbf{X}} \times \mathbf{R}^{\mathbf{X}} \rightarrow \mathbf{R}^{\mathbf{X}}$ of Hadamard multiplication ($\bigcirc = *$) over *range*(T) is given by

$$T(\mathbf{a} * \mathbf{b}) = \bigcirc'(\mathbf{a}_c, \mathbf{b}_c) = \sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{X}} (\mathbf{a}_c \oplus \mathbf{s})(\mathbf{x}) \cdot (\mathbf{b}_c \oplus \mathbf{s})(\mathbf{y}) \cdot T(\sigma[\vee(\mathbf{x},\mathbf{y})]), \tag{264}$$

where we assume that the maximum operation ($\vee$) is defined over $\mathbf{X}$.

Proof. From Lemma 6.3.2.8 and the givens, $\mathbf{a} = \sum_{y \in \mathbf{X}} (\mathbf{a} \oplus \mathbf{t})(\mathbf{y}) \cdot \sigma(\mathbf{y})$, and symmetrically for $\mathbf{b}$. Thus, the product

$$\mathbf{a} * \mathbf{b} = \sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{X}} (\mathbf{a} \oplus \mathbf{t})(\mathbf{x}) \cdot (\mathbf{b} \oplus \mathbf{t})(\mathbf{y}) \cdot \sigma(\mathbf{x}) \cdot \sigma(\mathbf{y}). \tag{265}$$

Since $\sigma(\mathbf{x}) \cdot \sigma(\mathbf{y}) = \sigma(\vee(\mathbf{x},\mathbf{y}))$, if maximum is defined over $\mathbf{X}$ and

$$\begin{aligned} T(\mathbf{a}) &= \sum_{y \in \mathbf{X}} T[(\mathbf{a} \oplus \mathbf{t})(\mathbf{y}) \cdot \sigma(\mathbf{y})] \\ &= \sum_{y \in \mathbf{X}} (\mathbf{a} \oplus \mathbf{t})(\mathbf{y}) \cdot T[\sigma(\mathbf{y})], \end{aligned} \tag{266}$$

then Equation 264 follows directly from the givens and Assumption 6.3.2.9. Thus, the theorem holds. □

6.3.2.11. Complexity. With slight manipulation, Equation 266 can be implemented efficiently on parallel architectures. For example, let $\mathbf{X} = \{1,2,...,n\}$, and let A be an n×n-processor mesh. If $\mathbf{a}$ is a $\mathbf{Z}_m$-valued image, then the y-th column of processing elements (PEs) can store the m partial products $f \cdot T(\sigma(x))(y), \forall f \in \mathbf{Z}_m$, where $x, y \in \mathbf{X}$. As a result, each PE stores $mn \cdot \lceil \log m \rceil$ bits. For example, if $n = 512$ and $m = 256$, as is usual, then each PE would store 1M bits. Given today's processors with large local memories, the requirement of 128KB storage per PE is not burdensome.

6.3.2.12. Remark. The preceding approach is advantageous, since a *discrete* linear transform $T(\mathbf{a})$ can be computed in log(n) addition delays, for a total cost of

$$C(n) = S(n) \cdot T(n) = n^2 \cdot \mathbf{O}(\log(n)) = \mathbf{O}(n^2 \cdot \log(n)). \tag{267}$$

C exceeds by a factor of n *space complexity* the cost of computing a fast version of T (e.g., the Fast Fourier Transform) on an n-processor butterfly architecture. However, in Equation 266, n is not limited to a power of 2. Additionally, if partial products are stored locally, then Equation 266 requires only addition operations, which can be integer additions. In contrast, the fast linear inner product transforms computed on butterfly architectures usually require real or complex local multiplications [72].

In practice, this approach would be useful for production transformation of images and signals where the image size and transform type are fixed, but time constraints predominate over space complexity. Given current advances in circuit miniaturization, it is not unreasonable to expect the availability of large, inexpensive SIMD meshes with large local memories in the foreseeable future. Thus, the preceding approach is realistic.

6.3.2.13. Data Security and information loss. The analogues of + and * over *range*(T_{1l}) do not compromise the security of the transform, which is insecure per the discussion of Sections 4.5–4.6. Such computations can suffer from information loss due to errors in $\mathbf{b}_c$ incurred by lossy or erroneous T as well as input errors in $\mathbf{a}$ or $\mathbf{b}$. Propagation of such errors is governed by the product rule given in Section 5.1.2.2. Assuming (for purposes of convenience) that the error operator δ is a linear function, if $\bigcirc'$ denotes the analogue over *range*(T_{1l}) of $\bigcirc = *$ (per Equation 264, then the error inherent in $\bigcirc'$ is given by

$$\epsilon_{\bigcirc'} = \epsilon_\Sigma\Big(\epsilon_\times\Big[\epsilon_T, \epsilon_\times\Big(\epsilon_{\bigoplus}(\epsilon_T[\epsilon_\mathbf{a}], \epsilon_\mathbf{s}), \epsilon_{\bigoplus}(\epsilon_T[\epsilon_\mathbf{b}], \epsilon_\mathbf{s})\Big)\Big]\Big)\Big), \tag{268}$$

where $\epsilon_T(\mathbf{a}) = \epsilon_{\bigoplus}(\epsilon_\mathbf{a}, \epsilon_\mathbf{r})$, and $\epsilon_\mathbf{r}$ denotes a representational error in the template $\mathbf{r}$, per Assumption 6.3.2.9.

6.3.2.14. Remark. There currently exists no general closed-form derivation of an analogue for nonlinear pointwise image operations (e.g., maximum or minimum) over the range space of a linear neighborhood transform T. Since neighborhood transforms are based upon noninvertible (many-to-one) mappings, it is difficult to predict the manner in which transform output represents individual image values. For example, in the simple case of pointwise transformations of form $T(x) = cx + d$, if (a) $c,x,d \geq 0$ or (b) $c,d < 0$ and $x \geq 0$, then the analogues of maximum and minimum are preserved in case a) and are the minimum and maximum operations, respectively, in case b). Alternatively, in the case of inner-product transformations such as the Fourier transform, analogues of nonlinear operations are currently not apparent.

We next discuss the derivation of global reduce operations over the range spaces of linear transforms.

6.3.3. Technique: Global Reduce Operations.

Since linear operations over linear transforms are themselves linear operations, the (linear) global reduce operation of summation can be formulated straightforwardly. However, the nonlinear operations are less obvious, and must be restricted to special cases, which we discuss as follows.

6.3.3.1. Assumption. Let the pointwise linear transformation $T_{1l}\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ be applied to an image $\mathbf{a} \in \mathbf{C}^{\mathbf{X}}$ to yield the transformed image $\mathbf{a}_c = (k_1 \cdot \mathbf{a}) + k_2$, where $k_1, k_2 \in \mathbf{C}$.

6.3.3.2. Lemma. Under constraint of Assumption 6.3.3.1, the dual (analogous) operation over *range*(T_{1t}) of the global reduce operation $\Gamma\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}$ of image summation is given by:

$$\Gamma'(\mathbf{a}_c) = (k_1 \cdot \Sigma\mathbf{a}) + k_2 \cdot |\mathbf{X}| \,. \tag{269}$$

Proof. The proof follows directly from Lemma 6.3.2.1 and the definition of a linear transform. Additionally, the offset k_2 is replicated $|\mathbf{X}|$ times, regardless of the values of $\mathbf{s}$. □

6.3.3.3. Lemma. Under constraint of Assumption 6.3.3.1, the dual (analogous) operation over *range*(T_{1t}) of the global reduce operation $\Gamma\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}$ of image maximum or minimum is given by:

$$\Gamma'(\mathbf{a}_c) = (k_1 \cdot \vee\mathbf{a}) + k_2 \quad \text{or} \quad \Gamma'(\mathbf{a}_c) = (k_1 \cdot \wedge\mathbf{a}) + k_2 \,. \tag{270}$$

Proof. The proof follows directly from Lemma 6.3.2.1 and the linearity property. □

6.3.3.4. Observation. The ease with which image sum can be retrieved from neighborhood (e.g., inner-product) transform representations varies with the transform. For example, in the case of the Fourier transform, the zero-order coefficient g_0 generally approximates the image mean. Thus, one can approximate the image sum $\Sigma\mathbf{a}$ by forming the product of g_0 and $|domain(\mathbf{a})|$. If $\mathbf{a}$ is block-encoded over $\mathbf{Y}$, then

$$\Sigma\mathbf{a} \approx \sum_{\mathbf{y}\in\mathbf{Y}} g_0(\mathbf{y}) \cdot kl, \tag{271}$$

where kl denotes blocksize.

6.3.4. Technique: Image-Template Operations.

We next discuss derivation of analogues of the image-template operations over the range space of linear transforms.

6.3.4.1. Observation. Theorem 6.3.2.10 covers the case of analogous image-template operations over $\mathbf{X}$, since $T(\mathbf{a}) = \mathbf{a} \oplus \mathbf{r}$, where $\mathbf{r} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$. That is, if $\mathbf{q} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$ and $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{q}) = \mathbf{a} \oplus \mathbf{q}$, then $T(\hat{\mathbf{a}}) = \mathbf{a} \oplus \mathbf{q} \oplus \mathbf{r}$ can be expressed as $\mathbf{a} \oplus \mathbf{p}$, where $\mathbf{p} = \mathbf{q} \oplus \mathbf{r}$. The following theorem is illustrative.

6.3.4.2. Assumption. Under constraint of Definition 6.3.2.7, Assumption 6.3.2.9, and Lemma 6.3.2.8, let images $\mathbf{a}, \mathbf{a}_c \in \mathbf{R}^{\mathbf{X}}$ such that the linear transformation $\mathbf{a}_c = T(\mathbf{a}) = \mathbf{a} \oplus \mathbf{r}$, with $\mathbf{r} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$. Let template $\mathbf{q} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$, such that $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{q}) = \mathbf{a} \oplus \mathbf{q}$. Let $\mathbf{s} = \mathbf{r}^{-1} \oplus \mathbf{q} \oplus \mathbf{t}$, where $\mathbf{t} = (-1\ \mathit{1})$, with the target pixel italicized.

6.3.4.3. Theorem. If Assumption 6.3.4.2 holds, then an analogue $\bigcirc' : \mathbf{R}^{\mathbf{X}} \times (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}} \to \mathbf{R}^{\mathbf{X}}$ of the image operation $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{q}) = \mathbf{a} \oplus \mathbf{q}$ over the range space of $T(\mathbf{a}) = \mathbf{a} \oplus \mathbf{r}$, where $\mathbf{r} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$, is given by

$$\bigcirc'(\mathbf{a}_c,\mathbf{s}) = T(\hat{\mathbf{a}}) = \sum_{\mathbf{y}\in\mathbf{X}} (\mathbf{a}_c \oplus \mathbf{s})(\mathbf{y}) \cdot T[\sigma(\mathbf{y})], \tag{272}$$

with σ defined previously.

Proof. Let Assumption 6.3.4.2 hold, and let $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{q}) = \mathbf{a} \oplus \mathbf{q}$ and $T(\mathbf{a}) = \mathbf{a} \oplus \mathbf{r}$, where $\mathbf{q},\mathbf{r} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$. From the proof of Theorem 6.3.2.10, we have $\mathbf{a} \oplus \mathbf{q} = \mathbf{a}_c \oplus \mathbf{r}^{-1} \oplus \mathbf{q}$, which leads to the following decomposition:

$$\begin{aligned} T(\hat{\mathbf{a}}) &= \sum_{y \in \mathbf{X}} T[(\mathbf{a}_c \oplus \mathbf{s})(y) \cdot \sigma(y)] \\ &= \sum_{y \in \mathbf{X}} (\mathbf{a}_c \oplus \mathbf{s})(y) \cdot T[\sigma(y)]. \end{aligned} \tag{273}$$

Thus, the theorem holds. □

We next examine the case of Class 2 transforms.

CHAPTER 7
CLASS 2 TRANSFORMATIONS

Recall (from Section 3.3) that the Class 2 transforms, denoted by T_2, modify only the grey levels of an image without transforming the image domain. Thus, the Class 2 transforms may be modelled as a greylevel function. The derivation of analogues and duals for the generalized Class 2 transform is discussed in Section 7.1. Pixel value compression via bit slicing is a common image compression method that has easily-derived analogues, as discussed in Section 7.2.

7.1. Generalized Class 2 Transform.

We present a generalization of the Class 2 transforms that can be used as a template for many of the commonly-used greyscale functions.

7.1.1. Technique: Pointwise and Global Reduce Operations.

We begin with a decomposition of T_2, then note several interesting properties of the Class 2 transforms.

7.1.1.1. Observation. Per Section 3.3, the generalized Class 2 transform can be decomposed as

$$ {}^{C}T_2 : \mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{X}} \quad \triangleq \Rightarrow \quad {}^{M}T_2 : \mathbf{F}^{\mathbf{m}} \to \mathbf{G} \quad \triangleq \Rightarrow \quad {}^{M}T_2 : \mathrm{f} \mapsto g(\mathrm{f}), \mathrm{f} \in \mathbf{F}, \tag{274} $$

where the greyscale function g is assumed to be a finite mapping.

We present the following definition and lemma, followed by several comments that cover the derivations given in the remaining sections of this chapter.

7.1.1.2. Definition. If an image value consists of a vector of n bits $(b_0, b_1, \ldots, b_{k-1}, \ldots, b_{n-k}, \ldots, b_{n-1})$, then the k-*th least significant bit* is denoted by b_{k-1} and the k-*th most significant bit* is denoted by b_{n-k}.

7.1.1.3. Lemma. Assume that Observation 7.1.1.1 holds, T_2 is based on a bijection g, and $\bigcirc : \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$ denotes an image operation. If $\bigcirc' : \mathbf{G} \rightarrow \mathbf{G}$ denotes the analogue of $\bigcirc$ over *range*(T_2), then

$$\bigcirc'(T_2(f), T_2(g)) = g \circ \bigcirc(f,g),\ f,g \in \mathbf{F}. \tag{275}$$

Proof. The proof follows directly from Observation 7.1.1.1 and the definition of homomorphism. □

7.1.1.4. Remark. As shown previously, we can implement Equation 7.1.1.3 without knowing the bijection g at each invocation of $\bigcirc'$, merely by encoding $\bigcirc'$ in a precomputed LUT. The customary LUT quantization error would hold for such an implementation.

7.1.1.5. Observation. Assume that Lemma 7.1.1.3 holds, where $\mathbf{F}$ is a finite subset of $\mathbf{R}$. Further assume that g is not a bijection, but preserves the k-th most significant bit of each pixel in T_2's input. It follows that T_2 is a lossy transform. Thus, $\bigcirc'$, an analogue of a binary greyscale operation $\bigcirc$ over *range*(T_2) can be applied to *range*(T_2) to yield the following ϵ-near approximation to $\bigcirc$:

$$\| \bigcirc'(T_2(f), T_2(g)) - \bigcirc(f,g) \| \leq 2\epsilon\ ,\ \forall f,g \in \mathbf{F}. \tag{276}$$

The upper limit of 2ϵ occurs in the case of addition, per the additive rule of error propagation.

7.1.1.6. Information Loss. In Observation 7.1.1.1, if $\bigcirc = +$, then where $2^k \leq \epsilon < 2^{k+1}$. This statement follows directly from the sum rule for error propagation (given in Chapter 5), and the fact that the truncation of k-1 high-order bits (with the modification of as many as n-k lower-order bits possible) can simulate division by a factor of $2^k \leq \epsilon < 2^{k+1}$.

Given an analogue of a pointwise binary operation the analogous global reduce function is readily derived, as follows.

7.1.1.7. Theorem. Let $T_2 : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ be applied to a source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ to yield an encoded image $\mathbf{a}_c$. The global reduce operation $\Gamma : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}$ has an analogue $\Gamma' : \mathbf{G}^{\mathbf{X}} \rightarrow \mathbf{G}$ over *range*(T_2), if the following conditions are satisfied:

(i) Γ is based upon the associative, commutative operation $\gamma : \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$;

(ii) γ has an analogue γ' over *range*(T_2); and

(iii) γ' is the associative, commutative function upon which Γ' is based.

Proof. The proof follows directly from the definitions of the global reduce operation and an analogous operation. □

We offer the following two examples as illustrations of an approximation to the global reduce operation, as well as a common pitfall in deriving analogues to Class 2 transforms.

7.1.1.8. Example. Suppose that T_2 is a mapping from a finite subset of the reals $\mathbf{F} = \{1.1, 2.2, 3.3, 4.4\}$ to an index set $\mathbf{G}$. Let $\mathbf{a} = (1.1, 3.3, 2.2)$, and let $T_2(x) = [2x]$, where [ ] denotes rounding to the nearest integer. Thus, $\mathbf{a}_c = T_2(\mathbf{a}) = (2, 7, 4)$, and $\Sigma\mathbf{a} = 6.6$ while $\Sigma\mathbf{b} = 12$. Since T_2 is a nonlinear approximation to a linear transform (i.e., $T(x) = 2x$), and can therefore be a lossy transform, the operation of addition does not have an analogue over *range*(T). As a result, $\Gamma'(T_2(\mathbf{a}))$, where $\Gamma' = \Sigma$, is an ϵ-near approximation to $T_2(\Sigma\mathbf{a})$, where $\epsilon > \frac{1}{3}(12 - (2 \cdot 6.6)) = 0.4$ if RMS error computation is employed.

Given that the predicted absolute error per pixel in T_2 is bounded by the interval [-0.5,0.5) or (-0.5,0.5] (i.e., the quantization error inherent in the rounding function), we

have that each invocation of γ' incurs an maximum absolute error of $1.0 = 0.5 + 0.5$ (i.e., in the case of addition). Since Γ' is comprised of two cascaded invocations of γ', the maximum absolute error is given by $1.5 = (0.5+0.5)+0.5$. In this instance, the computed error condition $\epsilon > 0.4$ is satisfied.

7.1.1.9. Example. Since we assume that T_2 is a finite homomorphism, it would appear that analogues could be formulated over non-numeric sets merely by indexing the value sets of T_2's input and output images (assuming that such value sets are unequal). However, this does not hold universally, especially in the case of modular arithmetic with heterogeneous moduli. For example, let $\mathbf{a},\mathbf{b} = (0,1,2)$, let $\mathbf{d}$ be defined in terms of its graph $G(\mathbf{d}) = \{(0,a),(1,b)\}$, and let $T_2(x) = \mathbf{d}(x \bmod 2)$. Thus, $\mathbf{a}_c = T_2(\mathbf{a}) = (a,b,a)$. Letting $\bigcirc = +_3$, we have $\mathbf{c} = \mathbf{a} +_3 \mathbf{b} = (0,0,0)$. It is easily verified that $\mathbf{c}_c = T_2(\mathbf{c}) = (a,a,a)$, since $\bigcirc(0,0) = 0$ and $\bigcirc(2,2) = 1$. However, the arguments 0 and 2 each have the same representation in *range*(T_2), namely, a. Thus, $\bigcirc = +_3$ has no analogous operation over the range space of this particular instance of T_2. Therefore, care must be exercised to inspect the underlying mappings that constitute the encoding, lest they comprise one or more of the cases for which analogues or duals cannot be derived by known means.

7.1.2. Image-Template Operations.

It follows from the preceding development as well as the development of Chapter 5, that a lossy encoding transform induces error propagation not only through the pointwise and global reduce transforms, but through the image-template operations as well. We previously discussed cases where the transform T_2 is implemented in terms of a finite bijection and is thus not lossy. Given such theory, it is relatively easy to formulate image-template operations over *range*(T_2).

7.1.2.1. Assumption. Let a source image $\mathbf{a} \in \mathsf{F}^{\mathbf{X}}$ be combined with a template $\mathbf{t} \in (\mathsf{F}^{\mathbf{X}})^{\mathbf{X}}$ to yield $\hat{\mathbf{a}} = \bigcirc(\mathbf{a},\mathbf{t}) = \mathbf{a} \textcircled{?} \mathbf{t}$. Let $T_2 \colon \mathsf{F}^{\mathbf{X}} \to \mathsf{G}^{\mathbf{X}}$ be based on a mapping $g \colon \mathsf{F} \to \mathsf{G}$, such that $\mathbf{a}_c = T_2(\mathbf{a})$, and let the template $\mathbf{s} \in (\mathsf{G}^{\mathbf{X}})^{\mathbf{X}}$ be defined by its weights as

$$\mathbf{s}_{\mathbf{y}}(\mathbf{x}) = T_2(\mathbf{t}_{\mathbf{y}}(\mathbf{x})),\ \mathbf{x},\mathbf{y} \in \mathbf{X}. \tag{277}$$

7.1.2.2. Observation. Under constraint of Assumption 7.1.2.1, if $g \colon \mathsf{F} \to \mathsf{G}$ is a bijection, then $\bigcirc' \colon \mathsf{G}^{\mathbf{X}} \times (\mathsf{G}^{\mathbf{X}})^{\mathbf{X}} \to \mathsf{G}^{\mathbf{X}}$, the analogue of $\bigcirc$ over *range*(T_2), is given by $\bigcirc'(\mathbf{a}_c,\mathbf{s}) = \mathbf{a}_c \textcircled{?}' \mathbf{s}$, where $\textcircled{?}' \colon \mathsf{G}^{\mathbf{X}} \times (\mathsf{G}^{\mathbf{X}})^{\mathbf{X}} \to \mathsf{G}^{\mathbf{X}}$ is expressed as

$$(\mathbf{a}_c \textcircled{?}' \mathbf{s})(\mathbf{y}) = \underset{\mathbf{x}\in\mathbf{X}}{\mathbf{\Gamma}'} \mathbf{a}_c(\mathbf{x}) \bigcirc' \mathbf{s}_{\mathbf{y}}(\mathbf{x}),\ \mathbf{y} \in \mathbf{X}, \tag{278}$$

with γ' (the basis operation of $\mathbf{\Gamma}'$) and $\bigcirc'$ denoting analogues of the operations γ and $\bigcirc$ that comprise $\textcircled{?}$.

7.1.2.3. Theorem. Under constraint of Assumption 7.1.2.1, if T_2 has an ϵ-near approximate inverse $T_2^* \colon \mathsf{G}^{\mathbf{X}} \to \mathsf{F}^{\mathbf{X}}$ that is based upon the ϵ'-near approximate inverse $g^* \colon \mathsf{G} \to \mathsf{F}$ of $g \colon \mathsf{F} \to \mathsf{G}$, then $\bigcirc' \colon \mathsf{G}^{\mathbf{X}} \times (\mathsf{G}^{\mathbf{X}})^{\mathbf{X}} \to \mathsf{G}^{\mathbf{X}}$, an analogue of $\bigcirc$ over *range*(T_2), would expressed as $\bigcirc'(\mathbf{a}_c,\mathbf{s}) = \mathbf{a}_c \textcircled{?}' \mathbf{s}$, where $\textcircled{?}' \colon \mathsf{G}^{\mathbf{X}} \times (\mathsf{G}^{\mathbf{X}})^{\mathbf{X}} \to \mathsf{G}^{\mathbf{X}}$ is given by

$$(\mathbf{a}_c \textcircled{?}' \mathbf{s})(\mathbf{y}) = \underset{\mathbf{x}\in\mathbf{X}}{\mathbf{\Gamma}'} \mathbf{a}_c(\mathbf{x}) \bigcirc' \mathbf{s}_{\mathbf{y}}(\mathbf{x}),\ \mathbf{y} \in \mathbf{X}, \tag{279}$$

with $\mathbf{\Gamma}'$ and $\bigcirc'$ denoting analogues of the operations $\mathbf{\Gamma}$ and $\bigcirc$ that constitute the operation $\textcircled{?}$.

Proof. The proof follows directly from the definition of homomorphism and an ϵ-near approximate inverse. □

7.2. Pixel Value Compression via Bit Slicing.

The technique of bit-plane elimination, otherwise called *bit slicing*, has long been known as a simple method of image compression. The customary method of bit slicing eliminates low-order bits, which may (in practice) represent noise. Other methods of bit slicing may remove higher-order bits that are thought not to have a significant effect on the visual representation of an image. However, the latter method requires intensive computation, in order to determine which bit planes are statistically less significant. We first describe the bit slicing transform, then derive analogues of basic IA operations over the transform's range space.

7.2.1. Basic Concepts.

We begin by analyzing a generalized formulation of the bit-slicing transform in terms of complexity, security, and information loss.

7.2.1.1. Assumption. Assume that $\mathbf{B} = \{0,1\}$ and that the greyscale function $g\colon \mathbf{B}^m \to \mathbf{B}^k$, where $k < m$.

7.2.1.2. Definition. The *bit slicing transform*, denoted at coarse granularity as ${}^{C}T_{2b}\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{X}}$, is decomposed as

$$ {}^{M}T_{2b}\colon (\mathbf{B}^m)^{\mathbf{X}} \to \left(\mathbf{B}^k\right)^{\mathbf{X}} \quad \triangleq \quad {}^{F}T_{2b}\colon f \to g(f), f \in \mathbf{B}^m . \tag{280} $$

It follows that $T_{2b} \sim g$.

7.2.1.3. Example. Let a source image $\mathbf{a} \in (\mathbf{B}^m)^{\mathbf{X}}$ whose image values are encoded as signed integers be compressed by right-shifting the bits of each image value by $l = m-k$ bits, to effect division by 2^l. Then, the basis function $g\colon \mathbf{B}^m \to \mathbf{B}^k$ of T_{2b} is implemented as $g(\mathbf{a}(\mathbf{x})) = \mathbf{a}(\mathbf{x}) \cdot 2^{-l}$, $\mathbf{x} \in \mathbf{X}$.

7.2.1.4. Compression Ratio. Since T_{2b} reduces the word width of its m-bit domain (determined by the function *siz*) by l bits, the compression ratio is given by

$$\mathrm{CR}(T_{2b}) = \frac{siz(\mathbf{B}^{\mathrm{m}})}{siz(\mathbf{B}^{\mathrm{m}-l})} = \frac{\mathrm{m}}{\mathrm{m}-l} = \frac{\mathrm{m}}{\mathrm{k}}. \tag{281}$$

7.2.1.5. Complexity. Assume that the image domain is denoted by $\mathbf{X}$. If T_{2b} is based upon $g\colon \mathbf{B}^{\mathrm{m}} \rightarrow \mathbf{B}^{\mathrm{k}}$, which is computed sequentially using an m-dimensional LUT, then the time complexity $\mathrm{T}(|\mathbf{X}|) = \mathbf{O}(|\mathbf{X}|)$, where the number of LUT invocations is independent of m. Alternatively, if N processors are employed to compute T_{2b} in $\mathbf{O}(|\mathbf{X}|/\mathrm{N})$ time, then each processor must have a datapath of width $\mathrm{V}(\mathrm{m},\mathrm{n})$ bits, which may be impractical for large m.

7.2.1.6. Data Security. For purposes of discussion, let T_{2b} implement a linear transformation (e.g., multiplication by the factor 2^{-l}). Thus, if T_{2b} is an encryptive homomorphism, then T_{2b} may be compromised by the method of linear-algebraic attack summarized in Ahituv [44].

It is interesting to note that a cipher based on nonlinear bit slicing would be more secure than T_{2b}. For example, a message characterized by the eight-bit word (*0* 1 *1* 0 *0* 1 0 1) could be encoded by removing the italicized bits, thereby yielding the 5–bit code (1 0 1 0 1). Since the input size is disclosed by T_{2b}, one merely has to determine m and k, then perform a brute-force attack requiring work $\mathrm{W}(\mathrm{m},\mathrm{k}) \leq |\mathbf{X}| \cdot \binom{\mathrm{m}}{\mathrm{k}}$ invocations of g.

For customary values of $16 \leq \mathrm{m} \leq 8$ bits and possible values of k, the attack is not burdensome over time scales of 30–60 minutes using current technology. For example, the required work is clearly bounded above by $|\mathbf{X}| \cdot (\mathrm{m}-1)! \approx 1.2 \cdot 10^{11} \cdot |\mathbf{X}|$ invocations of g. Let the input contain $|\mathbf{X}| \leq 3600$ m-bit symbols (i.e., 60 lines of 60 characters, or approximately one page of ASCII text). Assuming $|\mathbf{X}|$-fold parallelism and one machine cycle per invocation of g, one divides the preceding estimate by 1800 seconds to obtain a bandwidth requirement per processor of 66.7MHz, which is available commercially.

7.2.1.7. Information loss. Let *domain*(T_{2b}) be the m-bit integers, and let *range*(T_{2b}) be the k-bit integers, as implied by $g: \mathbb{B}^m \rightarrow \mathbb{B}^k$. Denoting entropy as H, the information loss $\triangle$H is given by:

$$\begin{aligned} \triangle H &= H(domain(T_{2b})) - H(range(T_{2b})) \\ &= siz(domain(T_{2b})) - siz(range(T_{2b})) \\ &= m - k. \end{aligned} \tag{282}$$

For example, if the word truncation portrayed in Example 7.2.1.3 is employed, then the information loss is clearly l bits per image pixel value, and $l \cdot |\mathbf{X}|$ bits per image.

7.2.1.8. Remark. Since T_{2b} is a lossy transform, the inverse transform T_{2b}^{-1} does not exist and T_{2b} is not an isomorphism. However, given the definition of T_{2b} contained in Example 7.2.1.3, it is possible to specify an ϵ-near approximation to T_{2b}^{-1}, which is denoted by T_{2b}^{*}: $(\mathbb{B}^k)^{\mathbf{X}} \rightarrow (\mathbb{B}^m)^{\mathbf{X}}$ and is based upon the mapping $g: \mathbb{B}^k \rightarrow \mathbb{B}^m$. The implementation is specified at fine granularity as $g(f) = f \cdot 2^l$, where l = m-k. In this case, the error associated with the image decompression operation T_{2b}^{*} is simply the information loss incurred by T_{2b}, which was determined in Equation 282.

For purposes of development, we assume that the bit slicing transform eliminates l lower-order bits, as shown in Example 7.2.1.3. This assumption is realistic, since it simulates clipping of low-order (noise) bits, and renders derivation of pointwise analogues over *range*(T_{2b}) more straightforward.

7.2.2. Technique: Pointwise Operations.

Pointwise operations over *range*(T_{2b}) are derived similarly to the pointwise operations over the range space of the generalized linear transform T_{1l}.

7.2.2.1. Lemma. Let source images $\mathbf{a}, \mathbf{b} \in (\mathbb{B}^m)^X$ be transformed by $T_{2b}: (\mathbb{B}^m)^X \rightarrow (\mathbb{B}^k)^X$, which is based on the function $g: \mathbb{B}^m \rightarrow \mathbb{B}^k$ (per Definition 7.2.1.2) to yield the compressed images $\mathbf{a}_c$ and $\mathbf{b}_c$. Further assume that T_{2b} has an ϵ-near approximation to the inverse transform, denoted by $T^*_{2b}: (\mathbb{B}^k)^X \rightarrow (\mathbb{B}^m)^X$. If the image operation $\bigcirc: \mathbb{B}^m \times \mathbb{B}^m \rightarrow \mathbb{B}^m$ has an analogue $\bigcirc': \mathbb{B}^k \times \mathbb{B}^k \rightarrow \mathbb{B}^k$ over *range*(T_{2b}), then the following statements hold:

(i) $\bigcirc' = \bigcirc$, e.g., if $\bigcirc = +$, then $\bigcirc' = +$; and

(ii) $\| T_{2b}(\bigcirc(\mathbf{a},\mathbf{b})) - T^*_{2b}(\bigcirc'(\mathbf{a}_c,\mathbf{b}_c)) \| < \epsilon'$, where $\epsilon' = \epsilon_{\bigcirc}(2^l, 2^l)$, with $\epsilon_{\bigcirc}$ the error function of $\bigcirc$.

Proof. Assuming that the givens hold, (i) follows from the definition of a linear transform, which T_{2b} is assumed to be. Statement (ii) follows from the definition of an ϵ-near approximation to the inverse transform, and (iii) follows from the definition of an error function of a binary operation, as well as the analysis of information loss given in Section 7.2.1.7. □

7.2.2.2. Remark. The ϵ-near approximation to the dual $\Omega: (\mathbb{B}^k)^X \times (\mathbb{B}^k)^X \rightarrow (\mathbb{B}^m)^X$ of the pointwise operation $\bigcirc$ is similarly derived, by setting $\Omega(\mathbf{a}_c, \mathbf{b}_c) = 2^l \cdot \bigcirc(\mathbf{a}_c, \mathbf{b}_c)$, which merely implements $\Omega = T^*_{2b} \circ \bigcirc'$, where the analogue $\bigcirc'$ was defined in the preceding lemma.

7.2.2.3. Complexity. Assume that $\bigcirc' = \bigcirc$, where $\bigcirc$ and $\bigcirc'$ compute over m-bit and k-bit integers, respectively. Further assume that the time complexities $T_{\bigcirc}(m) = \mathbf{O}(m)$ and $T_{\bigcirc'}(k) = \mathbf{O}(k)$. As shown in Section 4.2.3, the resultant speedup due to processing over *range*(T_{2b}) is given by $\eta = \mathbf{O}(m/k)$.

7.2.2.4. Data Security and Information Loss. For purposes of discussion, assume that T_{2b} is a lossy encryption with key l = m-k. If $\bigcirc' = \bigcirc$, then the analogue $\bigcirc'$ does not compromise the data security of T_{2b}, since no information is added to the inputs $\mathbf{a}_c$ and $\mathbf{b}_c$ of $\bigcirc'$. The preceding statement can be easily verified by noting that $\mathbf{b}_c$ cannot contain the l lower-order bits of $\mathbf{a}$ that are truncated by T_{2b}, since l lower-order bits of $\mathbf{b}$ are also truncated by T_{2b}. Furthermore, the approximate inverse T^*_{2b} does not compromise the data security of T_{2b}, since the replacement of l lower-order bits of $\mathbf{a}$ with zeroes (which is performed by the operation $T^*_{2b}(T_{2b}(\mathbf{a}))$) does not necessarily restore the information lost by T_{2b}.

However, attack upon T_{2b} is quite easy, for two reasons. First, T_{2b} preserves the plaintext domain size and ordering. Thus, if T_{2b} is available, the basis function g can be easily determined. Second, T_{2b} partially preserves the histogram of its input, which assists the cryptanalyst in discovering the plaintext, whether or not T_{2b} is available. In particular, if the adversary knows the frequency distribution of the l lower-order bits of each k-bit integer in $range(range(T_{2b}))$, then it is possible to reconstruct a probabilistic version of the plaintext histogram. With the domain ordering information that T_{2b} discloses, the adversary can construct a relaxation algorithm that guesses at n adjacent characters, based upon n-gram frequency tables. For purposes of efficiency, one would initially set n=2, then increment n until recognizable words appear in the decrypted plaintext. Thereafter, lexical- and relaxation-based methods would be combined to recognize probable words and their collocated candidate words [27].

7.2.3. Technique: Global Reduce Operations.

Given the pointwise analogue, formulating an analogue of the global reduce function is a straightforward process.

7.2.3.1. Theorem. Let source images $\mathbf{a}, \mathbf{b} \in (\mathbb{B}^m)^{\mathbf{X}}$ be transformed by $T_{2b}: (\mathbb{B}^m)^{\mathbf{X}} \rightarrow (\mathbb{B}^k)^{\mathbf{X}}$, which is based on the function $g: \mathbb{B}^m \rightarrow \mathbb{B}^k$ (per Definition 7.2.1.2), to yield the compressed images $\mathbf{a}_c$ and $\mathbf{b}_c$. Further assume that T_{2b} has an ϵ-near approximation to the inverse transform, denoted by $T^*_{2b}: (\mathbb{B}^k)^{\mathbf{X}} \rightarrow (\mathbb{B}^m)^{\mathbf{X}}$. If the global reduce operation $\Gamma: (\mathbb{B}^m)^{\mathbf{X}} \rightarrow \mathbb{B}^m$ has an analogue $\Gamma': (\mathbb{B}^k)^{\mathbf{X}} \rightarrow \mathbb{B}^k$ over *range*(T_{2b}), then the following statements hold:

(i) $\Gamma' = \Gamma$, e.g., if $\gamma = +$ is the basis function of Γ, then $\gamma' = +$ is the basis function of Γ'; and

(ii) $\| T_{2b}(\Gamma\mathbf{a}) - T^*_{2b}(\Gamma'(\mathbf{a}_c)) \| < \epsilon'$, where $\epsilon' = \epsilon_\gamma(2^l, 2^l)$ and ϵ_γ denotes the error function of γ.

Proof. The proof is outlined as follows. Assuming that the givens hold, (i) follows from Lemma 7.2.2.1 and the assumption that T_{2b} is a linear transform. Statement (ii) follows from the definition of an ϵ-near approximation to the inverse transform. □

7.2.3.2. Complexity and Data Security. Since $\Gamma' = \Gamma$, the complexity of Γ would initially appear to be preserved over *range*(T_{2b}). However, if the time complexity of γ is a function of the input word width, then it follows that the computational speedup incurred by processing over *range*(T_{2b}) using an analogue Γ' is given by

$$\eta_{(\Gamma,\Gamma')} = \frac{T_\Gamma(|\mathbf{X}|)}{T_{\Gamma'}(|\mathbf{X}|)} = \frac{|\mathbf{X}| \cdot T_\gamma(siz(\mathbf{F}))}{|\mathbf{X}| \cdot T_{\gamma'}(siz(\mathbf{G}))} = \frac{T_\gamma(m)}{T_\gamma(k)} = \frac{m}{k}. \tag{283}$$

The global reduce functions over *range*(T_{2b}) are not themselves secure, but do not compromise the transform's security. Note that the input image's arithmetic mean can be deduced from the image sum, and the image product discloses only the geometric mean of the input image. Similarly, the image maximum and minimum disclose no additional information about the input.

The dual operation is symmetrically defined, and has a similar error function. We next consider the pointwise image-template operations over *range*(T_{2b}).

7.2.4. Technique: Image-Template Operations.

Given the analogues of the pointwise and global reduce operations, the formulation of the image-template operation can proceed straightforwardly, per the methods of Section 3.4. In particular, since the bit-slicing transform affects only the values of the compressed image and not the image domain, the composition of γ and $\bigcirc$ to form $\textcircled{T}'$, the analogue of $\textcircled{T}$ over *range*(T_{2b}), is generally straightforward. However, if we apply T_{2b}^{-1} to the output of $\textcircled{T}'$, the result will be an approximation of the corresponding image operation, as discussed in the following sections.

7.2.4.1. Assumption. Let source image $\mathbf{a} \in F^{\mathbf{X}}$ and let template $\mathbf{t} \in (F^{\mathbf{X}})^{\mathbf{X}}$ be space-variant. Let $\mathbf{a}_c = T_{2b}(\mathbf{a})$, and let $\mathbf{s} \in (G^{\mathbf{X}})^{\mathbf{X}}$ be defined by its weights as $\mathbf{s}_{\mathbf{y}}(\mathbf{x}) = T_{2b}(\mathbf{t}_{\mathbf{y}}(\mathbf{x}))$, $\mathbf{x}, \mathbf{y} \in \mathbf{X}$. Let $\hat{\mathbf{a}} = \mathbf{a} \textcircled{T} \mathbf{t}$, and let γ and $\bigcirc$, the basis operations of $\textcircled{T}$, have analogues γ' and $\bigcirc'$ over *range*(T_{2b}), as described previously. Further assume that T_{2b} is an isomorphism with an ϵ-near approximation to its inverse denoted by T_{2b}^{*}.

7.2.4.2. Theorem. Under constraint of Assumption 7.2.4.1, $\textcircled{T}' : G^{\mathbf{X}} \times (G^{\mathbf{X}})^{\mathbf{X}} \rightarrow G^{\mathbf{X}}$, the analogue of $\textcircled{T}$ over *range*(T_{2b}) is given by:

$$(\mathbf{a}_c \textcircled{T}' \mathbf{s})(\mathbf{y}) = \underset{\mathbf{x} \in S(\mathbf{s}_{\mathbf{y}})}{\Gamma} \mathbf{a}_c(\mathbf{x}) \bigcirc \mathbf{s}_{\mathbf{y}}(\mathbf{x}),\ \mathbf{y} \in \mathbf{Y}, \tag{284}$$

such that

$$\left|\left| T_{2b}^{*}(\mathbf{a}_c \textcircled{T}' \mathbf{s}) - (\mathbf{a} \textcircled{T} \mathbf{t}) \right|\right| < \epsilon', \tag{285}$$

where $\epsilon' < \epsilon$, the error inherent in $T_{2b}^{*}(\mathbf{a})$.

Proof. Assuming that the givens hold, Equation 284 follows directly from the definition of $\textcircled{T}$ and the definition of a homomorphism. Similarly, Equation 285 follows from the definition of an ϵ-near approximation to the inverse transform. The condition $\epsilon' < \epsilon$ follows directly from the fact that $\textcircled{T}'$ introduces computational error in addition to that incurred by the operation $T_{2b}^{*}(\mathbf{a})$. □

7.2.4.3. Complexity. Under constraint of Assumption 7.2.4.1 and Theorem 7.2.4.2, if the analogous operations γ' and $\circ'$ incur identical computational cost as the corresponding image operations γ and $\circ$, then the image-template operation $\mathbf{a}_c \, \textcircled{T}' \mathbf{s}$ over $range(T_{2b})$ incurs computational cost identical to the operation $\mathbf{a} \, \textcircled{T} \, \mathbf{t}$. Since γ' and $\circ'$ compute over their operands in the same fashion as γ and $\circ$, it is reasonable to assume (for practical purposes) that $\gamma' = \gamma$ and $\circ' = \circ$. That is, if $\gamma = +$, then $\gamma' = +$, and likewise for $\circ'$. As a result, it would appear that no computational advantage accrues from processing over $range(T_{2b})$.

However, let us assume that $T_\gamma(m) = \mathbf{O}(m)$ and symmetrically for $\circ$ and k. For purposes of simplicity, let $|S(\mathbf{t})| \equiv |S(\mathbf{t}) - 1|$ and note that $|S(\mathbf{t_y})| = |S(\mathbf{s_y})|, \forall \mathbf{y} \in \mathbf{Y}$. If the value sets of T_{2b}'s domain and range spaces respectively consist of m- and k-bit numbers, then the speedup achieved by the image-template operation over $range(T_{2b})$ is given by:

$$\eta_{\textcircled{T}'}(\mathbf{t}, \mathbf{s}, m, k) = \frac{T_{\textcircled{T}}(\mathbf{t}, m)}{T_{\textcircled{T}'}(\mathbf{s}, k)} = \frac{\sum_{\mathbf{y} \in \mathbf{X}} (T_\gamma(m) \cdot |S(\mathbf{t_y})| - 1) + (T_\circ(m) \cdot |S(\mathbf{t_y})|)}{\sum_{\mathbf{y} \in \mathbf{X}} (T_{\gamma'}(k) \cdot |S(\mathbf{s_y})| - 1) + (T_{\circ'}(k) \cdot |S(\mathbf{s_y})|)}, \quad (286)$$

which becomes

$$\eta_{\textcircled{T}'}(\mathbf{t}, \mathbf{s}, m, k) = \frac{|\mathbf{X}| \cdot (T_\gamma(m) + T_\circ(m) \cdot |S(\mathbf{t_y})|)}{|\mathbf{X}| \cdot (T_\gamma(k) + T_\circ(k) \cdot |S(\mathbf{s_y})|)} = \mathbf{O}\left(\frac{m}{k}\right) = \mathbf{O}(CR). \quad (287)$$

Thus, in practice, one could process bit-sliced imagery with a speedup proportional to the compression ratio.

CHAPTER 8
CLASS 3 TRANSFORMATIONS

In this chapter, we overview Class 3 transforms of interest to image processing and cryptography. The *affine transform* (Section 8.1) is a component of fractal-based compression algorithms, whereas *pixel selection* (Section 8.2) is often employed in the reduction of sparse matrices. We conclude with the simple case of a *transpositional cipher* (Section 8.3).

8.1. Affine Transformation.

The affine transform derives from linear algebra and supports various transformations employed in computer graphics.

8.1.1. Basic Concepts.

8.1.1.1. Definition. Let domains $\mathbf{X}, \mathbf{Y} \subset \mathbf{R}^2$ and let the source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ be deformed by $T\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{Y}}$, which is based on the spatial transformation $g\colon \mathbf{Y} \to \mathbf{X}$, where each coordinate $\mathbf{x} = (x_1, x_2) \in \mathbf{X}$ is transformed as

$$\begin{aligned} \mathbf{y} = g(\mathbf{x}) &= (\mathrm{A} \cdot \mathbf{x} + \mathrm{B}) \\ &= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \end{aligned} \tag{288}$$

It is customary to call g an *affine transformation* and to express A as $\begin{pmatrix} r_1 \cos\theta_1 & r_1 \sin\theta_1 \\ -r_2 \sin\theta_2 & r_2 \cos\theta_2 \end{pmatrix}$, where (r_1, θ_1) and $(r_2, \theta_2 + \frac{\pi}{2})$ correspond to the points (a_{11}, a_{12}) and (a_{21}, a_{22}) expressed in polar form. Scaling is specified by r, rotation by θ, and the coefficient matrix B (reference Equation 288) specifies shift or spatial offset.

The image algebra formulation of T is given by

$$\mathbf{a}_c = T(\mathbf{a}) = \mathbf{a} \circ g \equiv \{(\mathbf{y}, \mathbf{a}(g(\mathbf{y}))) \colon \mathbf{y} \in \mathbf{Y}\}. \tag{289}$$

8.1.1.2. Observation. It is possible to redefine $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ in terms of a basis map $g: \mathbf{X} \rightarrow \mathbf{X}$ that incorporates restriction to $\mathbf{X}$. If $\mathbf{X}$ is finite, then the discrete affine transform is usually implemented in terms of an interpolation algorithm that improves the visual appearance of the transformed image. Unfortunately, such algorithms generally render the output of T a lossy approximation to the output of the corresponding affine transform over continuous $\mathbf{X}$. In Section 8.1.1.8, we present a general formulation for such losses. Additionally, we note that an affine transform can be space-variant, where A or B is a function of $\mathbf{x} \in \mathbf{X}$. However, we emphasize the space-invariant affine transform for purposes of simplicity.

8.1.1.3. Complexity. The space-invariant affine transform requires four multiplications and six additions per point in $\mathbf{X}$, yielding the time complexity

$$T_T(|\mathbf{X}|) = |\mathbf{X}| \cdot (4\,\Delta t_m + 6\,\Delta t_a). \tag{290}$$

If T is space-invariant, then A and B must be computed at each point in $\mathbf{X}$. For example, if the polar form given in Section 8.1.1.1 is employed, then four multiplications and invocations of the sine function are incurred per point in $\mathbf{X}$, in addition to the overhead shown in Equation 290.

8.1.1.4. Information loss. The average computational error inherent in Equation 288 is bounded as $|\epsilon_T| \leq 0.5 \cdot (q_1 + q_2)/2 = 0.25 \cdot (q_1 + q_2)$, where q_1 and q_2 denote the quantization intervals specific to the first and second coordinate of $\mathbf{X}$. This statement can be verified by noting that the maximum error in quantizing a real-valued number into an integer ($q = 1$) is $\pm q/2$. More precise quantification of information loss in a spatial affine transformation requires discussion of the errors inherent in an interpolation step that might be present. Rather than specialize our discussion to customary methods only, such as bilinear or bicubic-spline interpolation, we present a general discussion and formulation that will be useful in subsequent development.

8.1.1.5. Observation. Consider the case of image rotation in the absence of scaling. Let a pixel neighborhood $N(\mathbf{x})$, where $\mathbf{x} \in \mathbf{X}$ is rectangular, since $\mathbf{X} \subset \mathbf{R}^2$, be oriented arbitrarily at the center of a 3×3-pixel grid as shown in Figure 12a. It is well known that such a configuration cannot produce a rotated pixel that overlaps more than six pixels, inclusive of the central pixel, as shown in Figure 12b. The labelling scheme of Figure 12a intuitively suggests the following representation for error inherent in an affine transform of a rotated image.

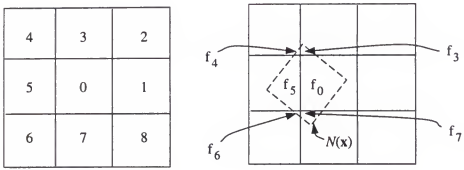


Figure 12. Pixel indexing and overlap scheme for information loss analysis of an affine transform (a) indexing scheme and (b) example of pixel overlap.

8.1.1.6. Definition. Let the discrete domain $\mathbf{X} \subset \mathbf{Z}_m^2$ and let the unscaled rotated image $\mathbf{a}_c = T(\mathbf{a})$, where T was given in Section 8.1.1.1. Given Figure 12, we can define $\mathbf{a}_c$ in terms of an n-pixel tesselation of a scaled, rotated, or shifted source pixel using a family of indexing functions $\mathbf{H} = \{h_i(\mathbf{y}) \in \mathbf{X} \colon i \in \mathbf{Z}_{n+1} \text{ and } \mathbf{y} \in \mathbf{Y}\}$. Additionally, we adopt the notational convenience that $f_i(h(\mathbf{y})) \equiv f(h_i(\mathbf{y}))$ since an implementation of the function f could pass i as an argument to h. Thus, we can write

$$\mathbf{a}_c \equiv \{(\mathbf{y}, [f_1(h(\mathbf{y})), f_2(h(\mathbf{y})), \ldots, f_n(h(\mathbf{y}))]) \colon \mathbf{y} \in \mathbf{Y}\}, \tag{291}$$

where f_i, i=1..n, denotes the fraction of the source pixel value $\mathbf{a}(h(\mathbf{y}))$ that is contained in $\mathbf{a}_c(\mathbf{y})$. For unscaled rotation restricted to $\mathbf{X}$, we can set $\mathbf{Y} = \mathbf{X}$ and the preceding convention reduces to n=8, per Figure 2.

8.1.1.7. Observation. Equation 291 facilitates specification of a discrete spatial transform T in terms of an interpolation that distributes fractions f_0 through f_8 of a pixel value spatially. By denoting the exact interpolation as $\widetilde{\mathbf{a}_c}$ and its attendant fractions as $\widetilde{\mathbf{f}}$, we are able to specify the interpolation error as shown in the following definition.

8.1.1.8. Definition. Given Definition 8.1.1.6 and the preceding observation, the error inherent in $\mathbf{a}_c = T(\mathbf{a})$ is given by

$$\begin{aligned} \epsilon_{\mathbf{a}_c}(\mathbf{y}) &= \mathbf{a}_c(\mathbf{y}) - \widetilde{\mathbf{a}_c}(\mathbf{y}) \\ &= \sum_{i=0}^{n} \mathbf{a}(h(\mathbf{y})) \cdot \left(\mathbf{f}_i(h(\mathbf{y})) - \widetilde{\mathbf{f}_i}(h(\mathbf{y})) \right), \mathbf{y} \in \mathbf{Y}, \end{aligned} \tag{292}$$

where $\widetilde{\mathbf{f}}$ was defined previously.

8.1.1.9. Remark. Implementationally, the approximation $\widetilde{\mathbf{a}_c}$ can be computed via line intersection and trapezoidal area formulae. Thus, Equation 292 provides an accurate theoretical expression for the error inherent in a space-invariant affine transformation T. Since $\mathbf{f}$ and $\widetilde{\mathbf{f}}$ vary spatially, this method is also useful for characterizing the error inherent in space-variant T.

8.1.1.10. Data Security. The space-invariant affine transformation shown in Equation 288 has little utility in data encryption, since it merely shifts, rotates, or scales an image, which does not render the image appearance obscure. As a result, correlation-based attack using multiple scaled and rotated exemplars is feasible. For example, one could employ high-bandwidth optical architectures based on Fourier-transform correlation [73]. In contrast, a space-variant transform can decorrelate the pixel ordering inherent in the source domain such that the transformed image may appear visually cryptic. As we have seen, such spatial transformation can approximate a transposition. Thus, the histogram of the transformed image approximates the histogram of the source image. As a result, an affine transform T is generally not considered useful for encryption, since T discloses the histogram of its input and might not obscure its input visually.

8.1.2. Derivation of Pointwise Analogues over *range*(T)

We begin by expressing the pointwise operations over $\mathbf{a}_c$, then analyze such operations in terms of the representation given in Equation 292.

8.1.2.1. Lemma. If Definition 8.1.1.1 holds and the system $X_{\bigcirc} = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{F}^{\mathbf{Y}}, \bigcirc'\})$ has the following salient components:

(i) $\bigcirc$ is a unary (binary) operation over $\mathbf{F}^{\mathbf{X}}$,

(ii) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ is a space-invariant affine transform, and

(iii) $\bigcirc'$ is a unary (binary) operation over $\mathbf{F}^{\mathbf{Y}}$,

then $\bigcirc' = \Theta(\bigcirc) = \bigcirc$.

Proof. The proof follows directly from the fact that an affine transform over continuous domains does not modify pixel values in T's input. The nearly trivial result is that $\bigcirc' = \bigcirc$. □

8.1.2.2. Remark. It follows from the preceding Lemma that the global-reduce operations and the image-template operation ⓣ are preserved over *range*(T), when T is continuous and space-invariant, provided that:

(i) T does not scale the source image when $\Gamma \in \{\Sigma, \Pi\}$, and

(ii) If $T(\mathbf{a})$ is restricted to $\mathbf{X}$, such restriction does not incur information loss, i.e., $\mathbf{X} \subset \mathbf{Y}$ is required.

Condition (i) is easily verified, since scaling by a factor of k would scale the image sum by k and the image product by $k^{|\mathbf{Y}|/|\mathbf{X}|}$. Condition (ii) prohibits cropping of $\mathbf{a}$ by T, which would prohibit Γ and ⓣ from having analogues, regardless of whether or not $\mathbf{X}$ is continuous.

We next discuss the more interesting case of analogues over the range of a discrete affine transform, using the representation given in Equation 291.

8.1.2.3. Assumption. Let Definition 8.1.1.1 hold, where $\mathbf{X}, \mathbf{Y} \in \mathbf{Z}_m^2$, and let Equation 292 describe the error inherent in an affine transform T.

8.1.2.4. Observation. Since $\mathbf{X}$ and $\mathbf{Y}$ are discrete and the error in T is nonzero, T is a lossy transformation. Thus, we say in practice that T has an ϵ-near approximation to its inverse, denoted by $T^*: \mathbf{F}^{\mathbf{Y}} \rightarrow \mathbf{F}^{\mathbf{X}}$ such that, for a given pair of images $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and $\mathbf{a}_c = T(\mathbf{a})$,

$$\| T^*(T(\mathbf{a})) - \mathbf{a} \| < \epsilon_{\mathbf{a}_c} + \delta_A , \tag{293}$$

where $\delta_A \in \mathbf{R}^+$ denotes implementational error (e.g., due to architecture-dependent precision).

8.1.2.5. Assumption. Let the system $V_{\bigcirc} = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{F}^{\mathbf{Y}}, \bigcirc'\}, T^*)$ have salient components that are specified as follows:

(i) $\bigcirc$ is a unary operation over $\mathbf{F}^{\mathbf{X}}$,

(ii) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ is an affine transform that has an ϵ-near approximation to its inverse given by $T^*: \mathbf{F}^{\mathbf{Y}} \rightarrow \mathbf{F}^{\mathbf{X}}$, such that $|\epsilon_{T^*}| = |\epsilon_T(\epsilon_{\mathbf{a}_c})| \leq 2\epsilon_{\mathbf{a}_c}$ (for purposes of convenience), where $\epsilon_{\mathbf{a}_c}$ is given in Equation 292; and

(iii) $\bigcirc'$ is a unary operation over $\mathbf{F}^{\mathbf{Y}}$.

8.1.2.6. Theorem. If Assumption 8.1.2.5 and Definitions 8.1.1.1, 8.1.1.6, and 8.1.1.8 hold and we have $\bigcirc' = \Theta(\bigcirc) = \bigcirc$, then

$$| T^*(\bigcirc'(T(\mathbf{a}))) - \mathbf{a} | \leq \epsilon_{\bigcirc}(\epsilon_{\mathbf{a}_c}) + 2\epsilon_{\mathbf{a}_c} , \tag{294}$$

where $\epsilon_{\bigcirc}$ denotes the error function of $\bigcirc$.

Proof. Assuming that $\bigcirc' = \bigcirc$, the proof follows directly from Lemma 8.1.2.1, the givens, and the law of composition of error functions (e.g., $\epsilon(f \circ g) = \epsilon_f \circ \epsilon_g$). The rightmost term in Equation 294 reflects the error inherent in T^*. □

An example of a discrete operation is given in the following theorem, which expresses $T(\mathbf{a})$ in terms of Equation 292.

8.1.2.7. Theorem. If Assumption 8.1.2.5 and Definitions 8.1.1.1, 8.1.1.6, and 8.1.1.8 hold, then

$$\hat{\mathbf{a}}_c(\mathbf{y}) = \bigcirc'(T(\mathbf{a})(\mathbf{y})) = \Theta(\bigcirc) = \sum_{i=1}^{n} \bigcirc(\mathbf{a}(h_i(\mathbf{y}))) \cdot \mathrm{f}_i(h(\mathbf{y})), \mathbf{y} \in \mathbf{Y}, \tag{295}$$

within the error bound of Theorem 8.1.2.6, where h was defined in Section 8.1.1.6.

Proof. Let the givens hold, and recall the commutativity diagram of Figure 1. Given the construction of $\mathbf{a}_c(\mathbf{y})$ shown in Definition 8.1.1.6, we obtain Equation 295 directly from the decomposition of $\mathbf{a}_c$. Since this decomposition varies from the exact decomposition $\tilde{\mathbf{a}}_c$ by the amount shown in Equation 292, the error bound of Theorem 8.1.2.6 holds. □

The case of an analogue of a binary pointwise operation is symmetrically defined.

8.1.2.8. Observation. In Theorem 8.1.2.7, if $\bigcirc$ is a linear operation, then

$$\begin{aligned} \hat{\mathbf{a}}_c(\mathbf{y}) = \bigcirc'(T(\mathbf{a})(\mathbf{y})) = \Theta(\bigcirc) &= \sum_{i=1}^{n} \bigcirc(\mathbf{a})(h_i(\mathbf{y})) \cdot \mathrm{f}_i(h(\mathbf{y})) \\ &= \sum_{i=1}^{n} \bigcirc(\mathbf{a}(h_i(\mathbf{y})) \cdot \mathrm{f}_i(h(\mathbf{y}))), \mathbf{y} \in \mathbf{Y}. \end{aligned} \tag{296}$$

This formulation may be slightly more efficient computationally on SIMD-parallel processor meshes that have linear I/O cost and a small operator set. For example, assume that $\bigcirc$ is implemented in terms of a LUT or instruction code that must be loaded prior to each invocation of $\bigcirc$. It would be more efficient to interleave (via pipelining) the LUT loading procedure while computing the product in Equation 296, then input the result to $\bigcirc$, versus loading the LUT, then computing $\bigcirc(\mathbf{a}(h_i(\mathbf{y})))$, then computing the product, as in Equation .

8.1.2.9. Complexity. Computing $\bigcirc'(\mathbf{a}_c) = \bigcirc(\mathbf{a}_c) \approx \bigcirc(\mathbf{a})$ requires only $|\mathbf{Y}|$ invocations of $\bigcirc$, which implies a speedup of $\eta = |\mathbf{X}|/|\mathbf{Y}| = CR_d$ that exceeds unity for minified images. Computation of Equation 295 requires $n \cdot |\mathbf{Y}|$ multiplications and invocations of $\bigcirc$, together with $(n-1) \cdot |\mathbf{Y}|$ additions. Thus, $\eta = |\mathbf{X}|/(n \cdot |\mathbf{Y}|)$, which implies that computational efficiency is not realized if $n > CR_d$.

Since T is insecure, $\bigcirc'(T(\mathbf{a}))$ is insecure. Thus, data security remains a moot issue. We next progress to a more detailed discussion of information loss.

8.1.2.10. Information loss. Since $\mathbf{a}_c = T(\mathbf{a})$ may be a scaled, rotated, or shifted version of $\mathbf{a}$ that is erroneous due to interpolation errors, we can further exploit the fractional representation f of Equation 292 by expressing the error in a given interpolation fraction f_i in terms of the error $\epsilon_{f_i} = f_i - \tilde{f}_i$, where $\tilde{f}_i$ denotes the exact value of f_i. The error in a unary pointwise operation $\bigcirc'$ formulated per Equation 295 is given by:

$$\begin{aligned} \epsilon_{\bigcirc'(\mathbf{a}_c(\mathbf{y}))} &= \epsilon_{\Sigma,n}(\epsilon_{\bigcirc}[\epsilon_X(\epsilon_a, \epsilon_{f_i})]) \\ &= \sum_{i=1}^{n} \epsilon_{\bigcirc}[\epsilon_X(\epsilon_a, \epsilon_{f_i})]. \end{aligned} \tag{297}$$

This provides us with an implementational expression that can be analyzed as follows.

8.1.2.11. Bit error rate. Note that $\epsilon_{\bigcirc'}$ predicted from Equation 297 is linear in f, which linearly comprises $\tilde{\mathbf{a}}_c$ and, therefore, $\epsilon_{\mathbf{a}_c}$. Thus, $\epsilon_{\bigcirc'}$ correlates with $\epsilon_{\mathbf{a}_c}$, and the likelihood $E(\epsilon_{\bigcirc'}) = E(\epsilon_{\mathbf{a}_c})$. As a result, $\bigcirc'$ does not compromise the bit-error rate of $\mathbf{a}_c$. As discussed previously, this issue is separate from degradation of bit-error rate by computational error due to the approximation inherent in T's interpolation scheme.

We next consider the simpler case of global reduce operations over *range*(T).

8.1.3. Global Reduce Operations

The derivation of analogous global reduce operations over the range space of an affine transform can be partitioned according to the additive or multiplicative form employed in error analysis.

8.1.3.1. Theorem. If Definition 8.1.1.1 holds and $\epsilon_{\mathbf{a}_c} \in \mathbf{R}^{\mathbf{Y}}$ is computed per Equation 292 then the CCS $X_{\mathbf{\Gamma}} = (\{\mathbf{F}^{\mathbf{X}}, \mathbf{F}; \mathbf{\Gamma}\}, \{T, I; \Theta\}, \{\mathbf{F}^{\mathbf{Y}}, \mathbf{F}; \mathbf{\Gamma}'\})$ has the following elements:

(i) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ is a space-invariant affine transform;

(ii) $I: \mathbf{F} \rightarrow \mathbf{F}$ is the identity transform; and

(iii) A dual of $\mathbf{\Gamma}$ is specified within the following error tolerance:

— If $\mathbf{\Gamma} = \Sigma$, $\vee$, or $\wedge$, then $\mathbf{\Gamma}' = \Theta(\mathbf{\Gamma}) = \mathbf{\Gamma}$ such that

$$|\mathbf{\Gamma}'(\mathbf{a}_c) - \mathbf{\Gamma}(\mathbf{a})| \leq \Sigma\epsilon_{\mathbf{a}_c}. \tag{298}$$

— If $\mathbf{\Gamma} = \Pi$, then $\mathbf{\Gamma}' = \Theta(\mathbf{\Gamma}) = \mathbf{\Gamma}$ such that

$$|\mathbf{\Gamma}'(\mathbf{a}_c) - \mathbf{\Gamma}(\mathbf{a})| \leq \mathbf{\Gamma}'(\mathbf{a}_c) \cdot \Sigma|\epsilon_{\mathbf{a}_c}/\mathbf{a}_c|. \tag{299}$$

Proof. The proof follows directly from the givens, as well as the sum and product laws of error propagation. We derive a dual rather than an analogue, since the latter would require that $T = I$, which is a trivial case. □

8.1.3.2. Observation. An alternative to Theorem 8.1.3.1 can be derived by setting $\bigcirc' = \gamma$, the basis function of $\mathbf{\Gamma}$, then composing $\bigcirc'$ (reference Equation 295) $|\mathbf{X}| - 1$ times over $\mathbf{a}_c$ to yield an approximation to $\mathbf{\Gamma}(\mathbf{a})$. The resultant error will be given by Equation 299.

8.1.3.3. Complexity. When Theorem 8.1.3.1 characterizes a dual $\mathbf{\Gamma}'$ of a global-reduce function $\mathbf{\Gamma}$, the work required by $\mathbf{\Gamma}'$ equals $|\mathbf{Y}| - 1$ invocations of γ, the basis function of $\mathbf{\Gamma}$. If the method of Observation 8.1.3.2 is employed in formulating γ', then $|\mathbf{Y}| - 1$ invocations of Equation 295 are required, which incurs $n \cdot (|\mathbf{Y}|-1)$ multiplications and $(n-1) \cdot (|\mathbf{Y}|-1)$ additions.

Information loss was discussed in Theorem 8.1.3.1. We next consider image-template operations over *range*(T).

8.1.4. Image-Template Operations.

We begin by assuming that the template is spatially invariant, for purposes of simplicity.

8.1.4.1. Assumption. Let Definition 8.1.1.1 hold, and let a source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ be convolved with a spatially invariant template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ as $\mathbf{b} = \mathbf{a} \textcircled{T} \mathbf{t}$. Let $\mathbf{t}$ be represented by its weight matrix $\mathbf{c} = \mathbf{t}_{\mathbf{x}}$, $\mathbf{x} \in \mathbf{X}$, which can be transformed by T to yield $\mathbf{c}_c \in \mathbf{F}^{\mathbf{Y}}$, the weight matrix of a template $\mathbf{s} \in (\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}$.

8.1.4.2. Theorem. If Assumption 8.1.4.1 holds and $\epsilon_{\mathbf{a}_c} \in \mathbf{R}^{\mathbf{Y}}$ is computed per Equation 292, then the CCS $X_{\textcircled{T}} = \left(\left\{\mathbf{F}^{\mathbf{X}}, (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}; \textcircled{T}\right\}, \{T, U; \Theta\}, \left\{\mathbf{F}^{\mathbf{Y}}, (\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}; \textcircled{T}'\right\}\right)$ has the following salient elements:

(i) $T: \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{Y}}$ is a space-invariant affine transform;

(ii) $U: (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \to (\mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$ produces $\mathbf{s} \in (\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}$ given $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, as discussed in Assumption 8.1.4.1; and

(iii) An analogue $\textcircled{T}' = \Theta(\textcircled{T}) = \textcircled{T}$ of $\textcircled{T}$ incurs computational error $\epsilon_{\textcircled{T}'}$ such that

$$\epsilon_{\textcircled{T}'} = |(\mathbf{a}_c \textcircled{T} \mathbf{s}) - T(\mathbf{a} \textcircled{T} \mathbf{t})| \leq 4\epsilon_{\mathbf{a}_c}. \tag{300}$$

Proof. The proof is outlined as follows. Assuming that the givens hold, observe that $\textcircled{T}' = \textcircled{T}$ combines each pixel in $\mathbf{a}_c$ with a pixel in $\mathbf{s}$ using the operation $\bigcirc: \mathbf{F} \times \mathbf{F} \to \mathbf{F}$. If $\bigcirc, \gamma \in \{+, \vee, \wedge\}$, then the additive law of error propagation applies and $\epsilon_{\bigcirc}, \epsilon_{\gamma} \leq 2 \cdot \epsilon_{\mathbf{a}_c}$. Here, we assume that $\epsilon_{\mathbf{a}_c}$ derives from information loss inherent in T, since $\mathbf{a}_c = T(\mathbf{a})$ and the template $\mathbf{s}$ has weight matrix $\mathbf{c}_c = T(\mathbf{c})$, where $\mathbf{c}$ denotes the weight matrix of $\mathbf{t}$. Thus, $\epsilon_{\textcircled{T}'} \leq 4 \cdot \epsilon_{\mathbf{a}_c}$, which yields Equation 300. If $\bigcirc' = \bigcirc = \cdot$ combines the values $\mathbf{a}_c(\mathbf{v})$ and $\mathbf{s}_{\mathbf{y}}(\mathbf{v})$, then the product law of error propagation applies and

$$\epsilon_{\bigcirc'} \leq \mathbf{a}_c(\mathbf{v}) \cdot \mathbf{s}_{\mathbf{y}}(\mathbf{v}) \cdot \left(\left|\frac{\epsilon_{\mathbf{a}_c}(\mathbf{v})}{\mathbf{a}_c(\mathbf{v})}\right| + \left|\frac{\epsilon_{\mathbf{s}_y}(\mathbf{v})}{\mathbf{s}_{\mathbf{y}}(\mathbf{v})}\right|\right), \mathbf{v} \in S(\mathbf{s}_{\mathbf{y}}), \mathbf{y} \in \mathbf{Y}, \tag{301}$$

where we assume $\epsilon_{\mathbf{s}} \approx \epsilon_{\mathbf{a}_c}$, as before. If we represent $\mathbf{a}_c(\mathbf{v})$ and $\mathbf{s}_{\mathbf{y}}(\mathbf{v})$ as p and q, then we can bound $\epsilon_{\bigcirc'} \leq 2pq \cdot \epsilon_{\mathbf{a}_c}$. Since we do not utilize the case where $\bigcirc, \gamma = \cdot$, we have only to consider $\textcircled{\scriptsize ?} \in \{\oplus, \textcircled{\scriptsize V}, \textcircled{\scriptsize \Lambda}\}$, where $\epsilon_{\textcircled{\scriptsize ?}'} \leq 2 \cdot \epsilon_{\bigcirc}$ due to the additive law of error propagation. Thus, Equation 300 holds, and the theorem holds. □

8.1.4.3. Observation. An alternative formulation of an analogue $\textcircled{\scriptsize ?}'$ of $\textcircled{\scriptsize ?}$ can be obtained by composing $\bigcirc'$, an analogue of $\bigcirc: \mathbf{F} \times \mathbf{F} \to \mathbf{F}$ over *range*(T), with γ', the basis function of Γ'. For example, let $\mathbf{a}, \mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ and let Theorem 8.1.2.7 hold for binary operations. Then, Equation 295 becomes

$$\begin{aligned} \bigcirc'(T(\mathbf{a})(\mathbf{y}), T(\mathbf{b})(\mathbf{y})) &= \Theta(\bigcirc) \\ &= \sum_{i=1}^{n} \bigcirc(\mathbf{a}(h_i(\mathbf{y})), \mathbf{b}(h_i(\mathbf{y}))) \cdot \mathrm{f}_i(h(\mathbf{y}),), \mathbf{y} \in \mathbf{Y}, \end{aligned} \tag{302}$$

where h and f were defined previously. Given $\mathbf{a}_c = T(\mathbf{a})$ and $\mathbf{s} = U(\mathbf{t})$ (from $X_{\textcircled{\scriptsize ?}}$ in Theorem 8.1.4.2) we can formulate $\textcircled{\scriptsize ?}'\colon \mathbf{G}^{\mathbf{Y}} \times (\mathbf{G}^{\mathbf{Y}})^{\mathbf{Y}} \to \mathbf{G}^{\mathbf{Y}}$ as follows

$$(\mathbf{a}_c \textcircled{\scriptsize ?}' \mathbf{s})(\mathbf{y}) = \underset{\mathbf{v} \in S(\mathbf{s}_{\mathbf{y}})}{\Gamma} \mathbf{a}_c(\mathbf{v}) \bigcirc' \mathbf{s}_{\mathbf{y}}(\mathbf{v}), \mathbf{y} \in \mathbf{Y}. \tag{303}$$

8.1.4.4. Remark. The work required by Equation 303 is given by $\sum_{\mathbf{y} \in \mathbf{Y}} |S(\mathbf{s}_{\mathbf{y}})|$ invocations of $\bigcirc'$ and $\sum_{\mathbf{y} \in \mathbf{Y}} (|S(\mathbf{s}_{\mathbf{y}})| - 1)$ invocations of γ'. However, each invocation of $\bigcirc'$ requires n multiplications and invocations of $\bigcirc: \mathbf{F} \times \mathbf{F} \to \mathbf{F}$, as well as n-1 additions, where n denotes the size of the interpolation neighborhood about a point in $\mathbf{Y}$. Thus, the actual work is given by

$$\begin{aligned} W_{\textcircled{\scriptsize ?}'}(|\mathbf{Y}|, n) = {} & n \cdot \sum_{\mathbf{y} \in \mathbf{Y}} |S(\mathbf{s}_{\mathbf{y}})| \cdot (W_{\times}(1) + W_{\bigcirc}(1)) \\ & + n \cdot \sum_{\mathbf{y} \in \mathbf{Y}} (|S(\mathbf{s}_{\mathbf{y}})| - 1) \cdot W_{+}(1). \end{aligned} \tag{304}$$

Note that the sequential time complexity $T_{\textcircled{\scriptsize ?}'}(|\mathbf{Y}|, n)$ is obtained by relabelling W in Equation 304 with T.

8.1.4.5. Information loss. From Chapter 5, we see that the information loss due to composition of functions γ' and $\bigcirc'$ is given by the composition of the error functions $\epsilon_{\gamma'}$ and $\epsilon_{\bigcirc'}$, which were defined previously. In particular, for each combination of k results of $\bigcirc'$ by the customary operations $\gamma' \in \{+, \vee, \wedge\}$, the absolute error is bounded above by $\epsilon_{\textcircled{\gamma}'} \leq k \cdot \epsilon_{\bigcirc'}$.

We next consider the case of pixel selection processes that preserve only a subset of the source image in T's output.

8.2. Spatial Transformation by Pixel Selection

We being with a general overview of the pixel selection technique, then progress to derivations of analogues or dual operations.

8.2.1. Basic Concepts.

Pixel selection is often employed in transform coding, where negligible transform coefficients are discarded, leaving a compressed transform representation.

8.2.1.1. Definition. Let $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and let $S \subset \mathbf{F}$ contain negligible values. Let $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ transform $\mathbf{a}$ to yield

$$\mathbf{a}_c \equiv \{(\mathbf{y},\mathbf{a}(\mathbf{x})) : \mathbf{y} = g(\mathbf{x},\mathbf{a}(\mathbf{x})), \mathbf{a}(\mathbf{x}) \notin S, \mathbf{x} \in \mathbf{X}\}, \tag{305}$$

where g is an indexing function that belongs to a family of functions $G : \mathbf{X} \times \mathbf{F} \rightarrow \mathbf{Y}$.

8.2.1.2. Example. Consider that T could select coefficients of a Fourier-transformed image to yield a list of non-negligible coefficients. Additionally, Equation 305 can be modified to recast T as a sparse-matrix reduction transform, by replacing the pixel value $\mathbf{a}(\mathbf{x})$ with the tuple $(\mathbf{x},\mathbf{a}(\mathbf{x}))$. Such development is given in Chapter 9.

8.2.1.3. Observation. Another way to describe T is to assume that the range space of $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ is reconfigured by T acting as a value dependent transposition based on $h: \mathbf{X} \times \mathbf{F} \rightarrow \mathbf{X}$, not as a value substitution of form $\mathbf{F} \rightarrow \mathbf{F}$. We can thus express T as a parameterized spatial transform

$$T(\mathbf{a}) = \mathbf{a} \circ h(\mathbf{a}), \tag{306}$$

where $h: \mathbf{F} \times \mathbf{Y} \rightarrow \mathbf{X}$ accepts a value in $\mathbf{a}$ and a location in $\mathbf{Y}$, and produces a source point in $\mathbf{X}$. Given this simplification of T, we can derive operations over *range*(T) by relabelling the domain points of $\mathbf{a}$, as follows.

8.2.2. Derivation of Analogues over *range*(T).

8.2.2.1. Theorem. If Observation 8.2.1.3 holds $\mathbf{X}, \mathbf{Y} \subset \mathbf{R}$, and the CCS $X_{\bigcirc} = (\{\mathbf{F}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{F}^{\mathbf{Y}}, \bigcirc'\})$ has the following salient components:

(i) $\bigcirc$ is a unary pointwise operation over $\mathbf{F}^{\mathbf{X}}$,

(ii) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ is a pixel selection transform; and

(iii) $\bigcirc'$ is a unary pointwise operation over $\mathbf{F}^{\mathbf{Y}}$;

then $\bigcirc'$ is given by $\bigcirc' = \bigcirc$, which incurs no additional information loss.

Proof. The proof follows directly from the givens and the fact that values in $\mathbf{a}$ are mapped directly to values in $\mathbf{a}_c$ without interpolation. Thus, the operation $\bigcirc' = \bigcirc$ introduces no computational error beyond that produced by applying $\bigcirc$ to a given value of $\mathbf{a}$. □

Analogues of binary operations over *range*(T) are symmetrically defined, if corresponding values in $\mathbf{a}$ and $\mathbf{b}$ are preserved in $\mathbf{a}_c$ and $\mathbf{b}_c$. Otherwise, when one operand of a binary pointwise operation is missing, it must be replaced with the identity element of that operation or with a value prespecified as characterizing the missing values.

8.2.2.2. Observation. Since values are eliminated from $\mathbf{a}$, image sum and product have neither an analogue nor a dual over *range*(T), unless one considers T to have an ϵ-near approximation to its inverse, denoted by $T^*: \mathbf{F}^{\mathbf{Y}} \rightarrow \mathbf{F}^{\mathbf{X}}$. For example, T^* could approximately reconstruct $\mathbf{a}$ from $T(\mathbf{a})$, given knowledge of the statistics of $\mathbf{a}$ and S, as well as the indexing function g.

8.2.2.3. Theorem. If Observation 8.2.2.2 holds and the image maximum (minimum) is not in the set of image values discarded by T, then the global reduce operations Γ of image maximum (minimum) are self-analogues over *range*(T).

Proof. For purposes of argument, assume that the image maximum (minimum), denoted by f, is in S. Then, f cannot be in *range*($\mathbf{a}_c$) and the corresponding analogous operation of image maximum (minimum) applied to $\mathbf{a}_c$ cannot satisfy the definition of homomorphism. Thus, the theorem holds. □

We next consider the case of the nonlinear image-template products, assuming that image maximum and minimum are preserved in *range*(T).

8.2.2.4. Assumption. Let $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ be a space-invariant template. Assume that $U: (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow (\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}$ transforms $\mathbf{t}$ into $\mathbf{s}$ by applying T to the weight matrix of $\mathbf{t}$ to yield the weight matrix of $\mathbf{s}$.

8.2.2.5. Theorem. If Assumption 8.2.2.4 holds, $\Gamma \in \{\vee, \wedge\}$, and the CCS $X_{\textcircled{T}} = \left(\left\{\mathbf{F}^{\mathbf{X}}, (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}; \textcircled{T}\right\}, \{T, U; \Theta\}, \left\{\mathbf{F}^{\mathbf{Y}}, (\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}; \textcircled{T}'\right\}\right)$ has the following elements:

(i) $T: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{Y}}$ is a pixel selection transform;

(ii) $U: (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \rightarrow (\mathbf{F}^{\mathbf{X}})^{\mathbf{Y}}$ produces $\mathbf{s} \in (\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}$ from $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$, per Assumption 8.2.2.4;

then $\textcircled{T}'$, an analogue of $\textcircled{T}$ over *range*(T), is given by $\textcircled{T}' = \textcircled{T}$.

Proof. Assuming the givens hold, $\mathbf{a}_c$ and $\mathbf{s}$ are spatial transforms of $\mathbf{a}$ and $\mathbf{t}$, with certain values removed. Recall that Theorems 8.2.2.1 and 8.2.2.3 respectively state that $\bigcirc : \mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$ and $\Gamma \in \{\vee, \wedge\}$ are self-analogues over *range*(T). Within the limits of information loss incurred by $\bigcirc'$ operating over reduced values in $\mathbf{a}$, we have $\textcircled{\Gamma}' = \textcircled{\Gamma}$, and the theorem holds. □

8.2.2.6. Example. Let a two-dimensional source image $\mathbf{a} \in [0,1]^{\mathbf{X}}$ (e.g., the bar chart of Figure 13a) be rotated by an affine transform T 30 degrees counterclockwise about the image domain's geometric center to form $\mathbf{a}_c$, shown in shown in Figure 13c. Let template $\mathbf{t} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$ (shown in Figure 13b) be rotated similarly to form template $\mathbf{s} \in (\mathbf{R}^{\mathbf{X}})^{\mathbf{X}}$, per Figure 13d. The locally averaged image $\mathbf{a} \oplus \mathbf{t}$ is shown in Figure 13e, with $T^*(\mathbf{a}_c \oplus \mathbf{s})$ shown in Figure 13f.

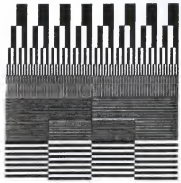
The error image that depicts blurring with the unrotated template $\mathbf{t}$ is shown in Figure 14a and is given by

$$\mathbf{e} = (T^*(\mathbf{a}_c \oplus \mathbf{t}))|_{\mathbf{X}} - (\mathbf{a} \oplus \mathbf{t}), \tag{307}$$

whereas blurring of $\mathbf{a}_c$ with the rotated template $\mathbf{s}$ generates less error, as shown in Figure 14b, which depicts

$$\mathbf{f} = (T^*(\mathbf{a}_c \oplus \mathbf{s}))|_{\mathbf{X}} - (\mathbf{a} \oplus \mathbf{t}). \tag{308}$$

The histogram of error intensities that corresponds to $\mathbf{e}$ (or $\mathbf{f}$) is given in Figure 14c (d).

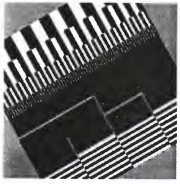


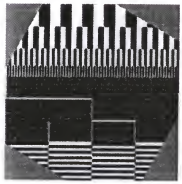
0	1	0
1	2	1
0	1	0

(a) (b)

0.261	0.802	0.261
0.802	1.662	0.802
0.261	0.802	0.261

(c) (d)

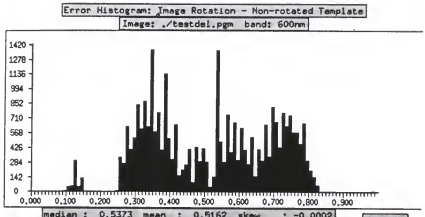




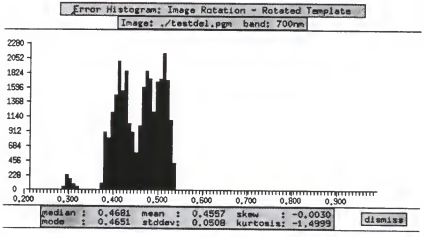
(e) (f)

Figure 13. Local averaging of a rotated Boolean bar chart using a rotated template: (a-b) source image $\mathbf{a}$ and template $\mathbf{t}$; (c-d) rotated image $\mathbf{a}_c = T(\mathbf{a})$ and template $\mathbf{s}$ formed by applying T to the weight matrix of $\mathbf{t}$, then restricting to the 3×3-pixel Moore configuration; (e) rotated locally-averaged image $\mathbf{a}_c \oplus \mathbf{s}$; (f) derotated image $T^*(\mathbf{a}_c \oplus \mathbf{s})$.

(a) (b)



(c)



(d)

Figure 14. Error images of local averaging using rotated and unrotated templates: (a-b) error images **e** (Equation 307) and **f** (Equation 308), taken from the central (uncropped) portion of *domain*(**a**); (c-d) histograms of error images **e** and **f**.

8.2.2.7. Remark. The preceding example demonstrates the utility of transforming the template as well as the image, when image-template operations are employed over the range space of an image transform. Note that the rotated template not only produces reduced error, but the errors are more closely clustered near the modes. When the derotated images (e.g., Figure 13f are normalized to the interval [0,1], the mean error about one standard deviation in Figure 14c (d) is given by 0.516 ± 0.176 (0.45 ± 0.05). Thus, using a rotated template reduces error more than threefold, from a standard deviation of 0.176 to 0.051.

8.3. Transpositional Ciphers.

The transpositional ciphers are a special case of the generalized Class 3 transform analyzed in Section 3.4. Since the transposition is a bijection f on the source domain $\mathbf{X}$, one need only apply f to the domain of an image operation $\bigcirc$ to produce the corresponding analogue over *range*(T). Salient theory is given in Theorem 3.4.2.11.

8.3.1. Properties of Transpositional Ciphers.

We begin with a few observations about transpositions and oracles.

8.3.1.1. Observation. In Section 4.4.4 we discussed oracles and in Theorem 4.4.4.20 we showed that Class 3 transforms preserve an ϵ-near approximation to the image histogram and sum. We also showed that if $\epsilon = 0$, then the image histogram and sum are preserved. Note that $\epsilon = 0$ for transpositional ciphers, which are usually considered as bijections over the source image domain.

8.3.1.2. Definition. A *transpositional cipher* $T\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ is based upon a spatial bijection $f\colon \mathbf{X} \to \mathbf{X}$, such that $\mathbf{a}_c = \mathbf{a} \circ f$, which was defined previously.

8.3.2. Cryptanalysis of Transpositional Ciphers.

We begin with a brief discussion of attack in the presence of the cipher, then progress to ciphertext-only attack.

8.3.2.1. Observation. If Definition 8.3.1.2 holds and the transpositional cipher T is available (with absence of the key understood), then plaintext attack is achieved as follows:

Step 1. Let $\mathbf{i} \equiv \{(\mathbf{x},\mathbf{x}) : \mathbf{x} \in \mathbf{X}\}$.

Step 2. Since values in $\mathbf{i}$ are unique, the graph $G(T) = \{(\mathbf{i},T(\mathbf{i}))\}$.

8.3.2.2. Remark. Recall the probabilistic oracle given in Definition 4.4.4.1, which yields a measure of probability that a given result of applying the oracle discloses a property of a specified cipher's input (plaintext). Additionally, Definition 4.4.4.1 discussed the assessment of error associated with such a result. Note that the preceding algorithm constitutes an oracle for plaintext attack on transpositional ciphers that yields the graph of the transposition with unitary probability and zero error. The property of the cipher that is preserved is the statement *preserves* $G(T) = \{(\mathbf{i},T(\mathbf{i}))\}$.

8.3.2.3. Observation. If Definition 8.3.1.2 holds, then the cardinality of the search space for enumerative ciphertext-only attack on a transpositional cipher is $|\mathbf{X}|!$. In principle, this is computationally prohibitive. In practice, the transposition is easily compromised by the method of histogram sorting and correlation, as follows.

8.3.2.4. Technique. Let Definition 8.3.1.2 hold and let a known plaintext corpus over the alphabet $\mathbf{F}$ have a histogram of i-grams $\mathbf{h}_\mathbf{i}$, where $\mathbf{i} > 1$. A transpositional cipher T is vulnerable to attack given ciphertext $\mathbf{a}_\mathbf{c}$only, via the following algorithm:

Step 1. Find the plaintext and ciphertext digram and trigram

$$(\mathbf{f},\mathbf{g}) = \mathit{domain}(\vee\mathbf{h}_2) \quad \text{and} \quad (\mathbf{f},\mathbf{g},\mathbf{h}) = \mathit{domain}\left(\vee\mathbf{h}_3|_{\{(\mathbf{f},\mathbf{g},\mathbf{h}):\,\mathbf{h}\in\mathbf{F}\}}\right), \tag{309}$$

where $f,g \in F$.

Step 2. Locate all occurrences of f, g, and h in $\mathbf{a}_c$ and collect their domain points in $\mathbf{X}$ in the sets F, G, and H, respectively.

Step 3. Construct the domain tuples $A(h) = \{(\mathbf{x},\mathbf{y}): \mathbf{x} \in F, \mathbf{y} \in G\}$ of all possible candidate pairs (f,g).

Step 4. For each a in A(h), collect in B(h) all possible triples $(\mathbf{x},\mathbf{y},\mathbf{z})$, where $\mathbf{z}$ is a point in H.

Step 5. Repeat Steps 1–4 for other trigrams beginning with letters other than h.

Step 6. A set of candidate digrams is found by the following intersection

$$C = \bigcap_{h \in F} \bigcap_{e \in F} A(h) \cap ([p_1(b),p_2(b)]), \quad b \in B(e). \tag{310}$$

By minimizing |C| through careful selection of trigrams in Step 5, one can force convergence of the algorithm.

8.3.2.5. Complexity. Given m digrams having a frequency of occurrence that is deemed significant, and n significant trigrams, the preceding algorithm requires $\mathbf{O}(mn|\mathbf{F}|^2)$ intersection operations, which is burdensome for large $\mathbf{F}$. Such comparisons are usually implemented as integer subtraction and comparison by zero, performed on the ciphertext coordinates in F, G, and H. If the plaintext is large, the overhead can be prohibitive. Thus, the following algorithm can be employed for purposes of efficiency.

8.3.2.6. Algorithm. In practice, the preceding technique can be implemented more efficiently using positional differences, as follows.

Step 1. *Find most frequent digram* (f,g) and compute positional differences between all f's and g's in $\mathbf{a}_c$.

Example. Let *AN* be the most frequent plaintext digram. Suppose a fragment of $\mathbf{a}_c$ is indexed as follows:

```
                Ciphertext Values:

                ...sdfjyAdfNthdoAjhfkNusdsAeiNt...
Positions:              |  |    |    |    |  |  |
  (units)       12345678901234567890123456789O

  (decade)                1          2         3
```

Step 2. *Extend candidate digrams:* Given all instances of *AN*, we find more frequent trigrams that begin with *AN*, and see if there are any consistent occurrences.

Example. For purposes of illustration, let *AND* be the most frequent plaintext trigram, followed by *ANY* and *ANT*. Continuing with our previous example, we have for the word *and*:

```
                Ciphertext Values:

                ...sdfjyAdfNthdoAjhfkNusdsAeiNt...
Positions:        #  *|#  |* # |     |  # |  |*
  (units)       12345678901234567890123456789O

  (decade)                1          2         3
```

where the positions of symbols *T* and *Y* are denoted by an asterisk and *D*'s positions are denoted by (#). We then construct the positional differences:

```
                Positional Differences (A-N,N-x)
 Trigram     1      2      3      4      5      6
-------  ------ ------ ------ ------ ------ ------
AND    3,-7   3,-2 3,3    3,13  5,-7 5,-17, ...
  ANY    3,-4   3,11  5,-14  5,1  3,-7  ---

  ANT    3,1    3,19  5,-9   5,9  3,-17 3,1
```

where *x* denotes *D*, *Y*, or *T*. The trigrams that repeat are *ANT*, with a difference of (3,1) that occurs twice.

Step 3. *Compare candidate trigrams:* Examine the trigram data to locate coincidences of the first two letters in any two trigrams (e.g,, the pairs (f,g) in our previous example). This increases the probability of finding a correct transpositional map for digrams that form the first two letters of the candidate trigram. This comparison

can also establish a set of candidate maps for the third letter(s) of trigrams that have more than one occurrence of a given set of positional differences. Repeat Steps 1–3 for different trigrams until a more complete candidate transposition emerges.

Example. Continuing with the preceding example, we see from the table in the preceding step that only one trigram (ANT) repeats is positional difference (3,1). Thus, we can assume that $A{**}NT$ is a reasonable first estimate of ANT in ciphertext, where (*) denotes ciphertext letters. We also note that in ANY and AND there is a preponderance of difference configurations of form (3,x). Thus, there is corroborative evidence that the transposition T separates adjacent letters A and N in plaintext via two intervening characters in ciphertext. For example, if $T(ANT)$ has positional differences (3,1), then NT has a unitary inter-symbol distance.

Step 4. *Trial inversion:* Given a candidate transposition, which will likely be a partial function over the message domain, invert the transposition and apply it to the ciphertext. This is possible, since T is a bijection. Look for common, recognizable words in the ciphertext, and attempt to fill in the blank symbols based on known plaintext statistics. Iterate Steps 1–4 until convergence is achieved.

8.3.2.7. Remark. The preceding algorithm constitutes an oracle that can determine if a given word (e.g., ANT) is in a given ciphertext. At convergence, the probability of finding a prespecified word is either one, or indeterminate. That is, without complete decryption, an oracle cannot accurately determine the probability with which a given word w would be found in ciphertext. As shown in Sinkov [27] and Patterson [30], one can employ Bayesian techniques to estimate such probabilities from n-gram distributions and knowledge of plaintext statistics. If the preceding algorithm does not converge, then no useful information is provided by the oracle. That is, the oracle's error measure would be maximized, and the probability would be indeterminate.

We next consider the Class 4 transforms.

CHAPTER 9
CLASS 4 TRANSFORMATIONS

This chapter contains derivations of analogues to the image algebra pointwise, global reduce, and image-template operations over the range spaces of several commonly-used Class 4 transforms. We begin with block encoding (Section 9.1), then progress to the processing of reduced sparse matrices (Section 9.2). Transform coding (Section 9.3), which employs matrix reduction, is instantiated in terms of JPEG compression (Section 9.4). Block truncation coding and VPIC are grouped together in Section 9.5, due to the similarity of their approaches. Having introduced the reader to codebook-based compression with VPIC, we then proceed to vector quantization (Section 9.6).

9.1. Block Encoding.

We discuss generalized block encoding as an introduction to the Class 4 transforms, which are primarily block-structured.

9.1.1. Basic Concepts.

Various transformations accept encoding blocks and index the blocks according to block, subblock or pixel properties. The indexing scheme is generally couched in terms of a *codebook*, which is a list of exemplars. Reconstruction (decompression) primarily emphasizes substitution of exemplars and, occasionally, normalization. Alternatively, a block can be decomposed into mean and variance measures associated with a zero-crossing bitmap. Decompression involved reconstructing the block using the mean perturbed by the bitmap, which is scaled by a measure of block variance.

9.1.1.1. Definition. Let $\mathbf{X}$ be an M×N-pixel array and let the source image $\mathbf{a} \in F^{\mathbf{X}}$ be tesselated by an indexing function h: $\mathbf{X} \rightarrow \mathbf{Y}$, which maps a point $\mathbf{x} \in \mathbf{X}$ in the y-th encoding block to $\mathbf{y}$. Additionally, Let h^*: $\mathbf{Y} \rightarrow 2^{\mathbf{X}}$ accept a point $\mathbf{y}$ in the compressed domain and return the corresponding encoding block domain. Further let f: $2^{\mathbf{X}} \rightarrow \mathbf{X}$ return a reference point in the encoding block (e.g., the block centroid or upper left-hand corner).

9.1.1.2. Definition. Given Definition 9.1.1.1, an *encoding block* $\mathbf{b}(\mathbf{y})$, $\mathbf{y} \in \mathbf{Y}$, is defined as

$$\mathbf{b}(\mathbf{y}) = \mathbf{a}|_{h^*(\mathbf{y})}. \tag{311}$$

9.1.1.3. Algorithm. If Definitions 9.1.1.1 and 9.1.1.2 hold, then for each source image block $\mathbf{b}(\mathbf{y})$, $\mathbf{y} \in \mathbf{Y}$, a codebook-based block encoding transform T is computed as follows:

Step 1. Determine the block properties $\mathbf{Q} = \{q_1, q_2, ..., q_n\}$.

Step 2. From the block properties determine a block index i that corresponds to a best-match exemplar pattern in a *codebook* $\mathbf{c}$ (a list of block exemplars), when the match satisfies a prespecified error criterion ϵ.

Step 3. Depict the source block $\mathbf{b}(\mathbf{y})$ in the compressed image as

$$\mathbf{a}_c(\mathbf{y}) = \left(\Upsilon_{\lambda \in D \subset \Lambda} p_\lambda(\mathbf{y}), \mathrm{i}(\mathbf{y}) \right), \tag{312}$$

where Λ indexes $\mathbf{Q}$ and Υ denotes a tuple constructor.

Decompression involves reconstruction of each encoding block as

$$\tilde{\mathbf{b}} = f(g(p_1(\mathbf{a}_c)), \mathbf{c}(p_2(\mathbf{a}_c))), \tag{313}$$

where the second argument of f is a codebook exemplar. In cases where a codebook is not employed, $p_2(\mathbf{a}_c)$ is null.

9.1.1.4. Remark. An additional block encoding transform, called *transform coding*, applies an operation such as the DCT to a block, then selects significant coefficients for further compression based on prespecified criteria. Since only a partial block transform is retained, the decompressed image only approximates the source image.

9.1.1.5. Observation. In order for T to be compressive, the set D and the bounds on i must be chosen such that

$$|domain(\mathbf{a}_c)| \cdot siz(range(\mathbf{a}_c)) < |domain(\mathbf{a}_c)| \cdot siz(range(\mathbf{a}_c)). \tag{314}$$

This implies a priori knowledge of image properties and optimal or near-optimal choice of Q for efficient compression. However, the properties that yield minimum entropy in the compressed image do not necessarily facilitate operations over *range*(T).

The derivation of pointwise, global reduce, and image-template operations over block-structured compression transforms was discussed in Section 3.4. We next consider the processing of reduced sparse matrices.

9.2. Sparse Matrix Processing.

Sparse matrix reduction is fundamental to the JPEG transform and to the majority of transform coding techniques. Since matrix reduction generally preserves the relationship between selected image pixel values and their domain points, a given image operation is (at a high level) generally a self-analogue over a given reduced-matrix data structure. However, the image operation must be modified slightly (at a low level) to accept the reduced-matrix format.

9.2.1. Basic Concepts.

We begin by defining the sparse matrix reduction transform T_{SM}, then form the basis for derivation of operations over *range*(T_{SM}).

9.2.1.1. Definition. Given a source image $\mathbf{a} \in \mathsf{F}^{\mathbf{X}}$ and a set of negligible values $S \subset \mathsf{F}$, let $\mathbf{a}$ be transformed by the *sparse matrix reduction transform* $T_{SM}: \mathsf{F}^{\mathbf{X}} \rightarrow (\mathbf{X} \times \mathsf{F})^{\mathbf{Y}}$ to yield a compressed image:

$$\mathbf{a}_c \equiv \{(\mathbf{y},(\mathbf{x},\mathbf{a}(\mathbf{x}))) : \mathbf{y} = h(\mathbf{x},\mathbf{a}(\mathbf{x})) \text{ and } \mathbf{a}(\mathbf{x}) \notin S, \mathbf{x} \in \mathbf{X}\}, \tag{315}$$

where h is derived from the family of indexing functions $H : \mathbf{X} \times \mathsf{F} \rightarrow \mathbf{Y}$. We call $\mathbf{a}_c$ a *reduced-matrix representation of* $\mathbf{a}$.

9.2.1.2. Complexity. The computation of T_{SM} clearly requires $\mathbf{O}(|\mathbf{X}|)$ comparison operations to determine if $\mathbf{a}(\mathbf{x})$ is in S, together with $\mathbf{O}(|\mathbf{Y}|)$ I/O operations to enlist the non-negligible values in the graph representation of $\mathbf{a}$ (similar to Equation 315) together with their domain points.

9.2.1.3. Information loss. Since values in $\mathbf{a}$ are discarded, T_{SM} is a lossy transformation. In particular, if $\mathbf{h} \in \mathsf{N}^{\mathsf{F}}$ denotes the histogram of $\mathbf{a}$, then the error with which $\mathbf{a}_c$ represents $\mathbf{a}$ is computed as follows. First, let $\tilde{\mathbf{a}} \in (\mathsf{F} \setminus S)^{\mathbf{X}}$ denote an image reconstructed from $\mathbf{a}_c$. Letting the error image $\mathbf{e} = \tilde{\mathbf{a}} - \mathbf{a}$, we can compute the mean error per pixel $\overline{\epsilon_{SM}} = \mathbf{e}/|\mathbf{X}|$ from $\mathbf{h}$ as follows:

$$\overline{\epsilon_{SM}}(S,\mathbf{a}) = \frac{1}{|\mathbf{X}|} \cdot \sum_{f \in S} \mathbf{h}(\mathbf{a})(f) \cdot f, \tag{316}$$

since it is precisely the values in S that are discarded.

9.2.2. Derivation of Operations over *range*(T_{SM}).

9.2.2.1. Theorem. If T_{SM} is given by Definition 9.2.1.1, then unary image operations over *domain*(T_{SM}) are self-analogues over *range*(T_{SM}), with slight refinement to accomodate data structures in *range*(T_{SM}).

Proof. Let the givens hold and let $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ with $\bigcirc: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$. If $T_{SM}: \mathbf{F}^{\mathbf{X}} \rightarrow (\mathbf{X} \times \mathbf{F})^{\mathbf{Y}}$, then let the refinement

$$\bigcirc'(p_2[T_{SM}(\mathbf{x},\mathbf{a}(\mathbf{x}))]) = \bigcirc(\mathbf{a}(\mathbf{x})) \tag{317}$$

define an analogue of $\bigcirc$ over *range*(T_{SM}). Thus, $\bigcirc$ fulfills the definition of self-analogue in the context of the preceding refinement.

□

9.2.2.2. Theorem. If T_{SM} is given by Definition 9.2.1.1, then binary image operations over *domain*(T_{SM}) are self-analogues over *range*(T_{SM}), provided that one of the following conditions holds:

(i) Let an image operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ that has scalar arguments denoted by $\mathrm{f}, \mathrm{g} \in \mathbf{F}$ be applied to images $\mathbf{a}$ and $\mathbf{b}$. Assume that $(\mathbf{x}, \mathrm{f})$, $\mathbf{x} \in \mathbf{X}$ is the pixel value of $\mathbf{a}_c(\mathbf{y})$, where $\mathbf{y} \in \mathbf{Y}$ is a domain point of $\mathbf{a}_c = T(\mathbf{a})$. Symmetrically, let $(\mathbf{x},\mathrm{g}) = \mathbf{b}_c(\mathbf{z})$, where $\mathbf{z} \in \mathbf{Y}$. Then,

$$\bigcirc'(\mathbf{a}_c(\mathbf{y}), \mathbf{b}_c(\mathbf{z})) = \bigcirc(p_2(\mathbf{a}_c(\mathbf{y})), p_2(\mathbf{b}_c(\mathbf{z}))) = \bigcirc(\mathrm{f},\mathrm{g}). \tag{318}$$

(ii) If $(\mathbf{x}, \mathrm{f}) \in$ *range*($\mathbf{a}_c$) has no corresponding $(\mathbf{x},\mathrm{g})$ in *range*($\mathbf{b}_c$), then

$$\bigcirc'(\mathbf{a}_c(\mathbf{y}), \mathbf{b}_c(\mathbf{z})) = \bigcirc(p_2(\mathbf{a}_c(\mathbf{y})),\mathrm{e}) = \bigcirc(\mathrm{f},\mathrm{e}) = \mathrm{f}, \tag{319}$$

where e denotes the identity of $\bigcirc$.

Otherwise, $\bigcirc'(\mathbf{a}_c(\mathbf{y}), \mathbf{b}_c(\mathbf{z}))$ is undefined.

Proof. If one considers Equations 318 and 319 to be refinements of $\bigcirc$, then the proof is symmetric to that of the preceding theorem.

□

9.2.2.3. Complexity. If $\mathbf{Y}$ is indexed by h per Definition 9.2.1.1, then h has a dual formulation $h^*: \mathbf{Y} \rightarrow \mathbf{X}$, since h is one-to-one and onto. The latter map can be implemented in a lookup table for constant-time execution or in a mintree, which requires a minimum of $\lceil \log(|\mathbf{Y}|) \rceil$ comparisons. Unfortunately, the use of h^* in searching for a source point in $\mathbf{X}$ generally requires search over $\mathbf{Y}$. If Case (i) of Theorem 9.2.2.2 pertains, then searching for corresponding source points could require $\mathbf{O}(|\mathbf{Y}|^2)$ overhead in the worst case.

Alternatively, h^* can be inverted to yield $\overline{h}: \mathbf{X} \rightarrow \mathbf{Y} \cup \{\perp\}$, which can return the location $\mathbf{y} \in \mathbf{Y}$ (or the undefined symbol $\perp$) if a source point is (is not) preserved in a given reduced matrix representation. The complexity of $\overline{h}$ is $\mathbf{O}(|\mathbf{X}|)$ (or $\mathbf{O}(|\mathbf{Y}|)$ if only those source points in the reduced matrix) are indexed. Symmetrically, the minimum search overhead for a tree-based implementation of $\overline{h}$ would be $\mathbf{O}(\log(|\mathbf{X}|))$ or $\mathbf{O}(\log(|\mathbf{Y}|))$.

9.2.2.4. Information loss. We consider only Theorem 9.2.2.2(i), since (ii) has unpredictable error due to the replacement of $\bigcirc(f,g)$ with f when g is undefined or unavailable in the reduced matrix. In (i), $\bigcirc'$ is computed only when the corresponding operation $\bigcirc$ has operands f and g present. Given a set of negligible values S, if either $f \in S$ or $g \in S$, then $\bigcirc(f,g)$ is not computed.

Let $\mathbf{a}$ and $\mathbf{b}$ be two source images transformed by T_{SM} to yield $\mathbf{a}_c$ and $\mathbf{b}_c$, where $\mathbf{h}$ denotes the histogram

$$\mathbf{h}(f,g) = \Sigma \chi_f(\mathbf{a}) \cdot \chi_g(\mathbf{b}),\ f,g \in \mathit{range}(\mathbf{a}) \cup \mathit{range}(\mathbf{b}). \tag{320}$$

Per Equation 316, the information loss per pixel in $\bigcirc'(\mathbf{a}_c,\mathbf{b}_c)$ due to value elimination in T_{SM} is given by

$$\overline{\epsilon_{\bigcirc'}} = \frac{1}{|\mathbf{X}|} \cdot \sum_{f \in S} \sum_{g \in S} \mathbf{h}(f,g) \cdot \bigcirc(f,g). \tag{321}$$

9.2.2.5. Observation. Assume that $\bigcirc' = \bigcirc$ at a high level (i.e., at coarse granularity) such that a slight refinement in $\bigcirc$'s data access process(es) is required to form $\bigcirc'$. Since a binary operation is employed in global-reduce and image-template operations, Theorem 9.2.2.2 holds in such cases. As a result, if corresponding source points are found in two reduced matrices (images), then $\bigcirc$ can be applied to yield $\bigcirc'$. Otherwise, e (the identity of $\bigcirc$) may be introduced to let $\bigcirc(f,e) = f$ (per Equation 319), or we assume that $\bigcirc$ is undefined.

We next consider transform coding, an application of T_{SM}.

9.3. Transform Coding.

Transform coding applies an image transformation to a source image (usually tesselated blockwise), followed by sparse matrix reduction. For purposes of discussion, transform coefficients are treated in principle as a sparse matrix and are selected in practice based on various criteria (entropy, coefficient magnitude, preselected feature attributes, etc.) In the preceding section, we showed that sparse matrices facilitate self-analogues of image operations with slight refinement to accomodate data structures. Thus, derivation of an analogue of an image operation $\bigcirc$ over transform-coded imagery is relatively straightforward, provided that the $\bigcirc$-property of the given transform is known.

9.3.1. Basic Concepts.

We begin with a general definition of transform coding that is simplified for purposes of clarity.

9.3.1.1. Definition. Let $\mathbf{X}$ be a finite discrete domain and let $\mathbf{a} \in \mathsf{F}^{\mathbf{X}}$ be transformed by $R\colon \mathsf{F}^{\mathbf{X}} \rightarrow \mathsf{H}^{\mathbf{W}}$, where $range(R) = domain(R)$ is possible. Transform coding, denoted by $T_{TC}\colon \mathsf{F}^{\mathbf{X}} \rightarrow (\mathrm{W} \times \mathsf{H})^{\mathbf{Y}}$ is given by:

$$T_{TC}(\mathbf{a}) = T_{SM}(R(\mathbf{a}), \mathrm{S}), \tag{322}$$

where S denotes one or more criteria (e.g., a threshold) that defines negligible values in the sparse matrix reduction transform T_{SM}.

9.3.1.2. Complexity. The work required by T_{TC} is given by

$$\mathrm{W_{TC}}(\mathbf{a}) = \mathrm{W}_R(\mathbf{a}) + \mathrm{W_{SM}}(\mathbf{H}, \mathbf{W}), \tag{323}$$

where $\mathrm{W_{SM}}$ was discussed previously, and we assume that the work requirement of image transformation (W_R) is known a priori.

9.3.1.3. Information loss. T_{TC} incurs information loss that can be expressed in terms of the error function

$$\epsilon_{\mathrm{TC}}(\mathbf{a}) = \epsilon_{\mathrm{SM}}[\epsilon_R(\epsilon_{\mathbf{a}})], \tag{324}$$

where ϵ_{SM} was given previously, and we assume that ϵ_R is known a priori.

9.3.2. Deriving Operations over *range*(T_{TC}).

Derivation of operations over *range*(T_{TC}) can be straightforward, requiring only a knowledge of properties of the image transformation R, in addition to previously-developed theory.

9.3.2.1. Theorem. Let Definition 9.3.1.1 hold and let Ω denote an analogue over *range*(R) of the image operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$. If Ω' denotes an analogue of Ω over *range*(T_{SM}), then $\bigcirc'$, an analogue of $\bigcirc$ over *range*(T_{TC}), is defined in terms of images $\mathbf{a}, \mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ as

$$\bigcirc'(T_{\mathrm{TC}}(\mathbf{a}), T_{\mathrm{TC}}(\mathbf{b})) = T_{\mathrm{TC}}(\bigcirc(\mathbf{a},\mathbf{b})) = \Omega'(R(\mathbf{a}), R(\mathbf{b})). \tag{325}$$

Proof. If the givens hold, then by definition of an analogue, $\Omega(R(\mathbf{a}),R(\mathbf{b})) \in \mathbf{H}^{\mathbf{W}}$. Similarly, $\Omega'(R(\mathbf{a}),R(\mathbf{b})) \in (\mathbf{W} \times \mathbf{H})^{\mathbf{Y}} = \mathit{range}(T_{\mathrm{TC}})$, and the theorem holds.

□

9.3.2.2. Remark. Since it is couched in terms of a binary operation, the preceding theorem can be thought of as generalizing unary and binary pointwise, global reduce, and image template operations over *range*(T_{TC}).

9.3.2.3. Information loss. Assume that Ω' is an analogue over *range*(T_{SM}) of the reduced-matrix operation Ω that incurs error $\epsilon_{\Omega'}$. Let an operation over *range*(T_{SM}) be a self-analogue, with slight refinement to accomodate the reduced matrix data structure, and let $\mathbf{a}, \mathbf{b} \in \mathbf{F}^{\mathbf{X}}$ with $\mathbf{a}_c = T_{\mathrm{TC}}(\mathbf{a})$ and symmetrically for $\mathbf{b}$. Then, $\bigcirc'$ incurs information loss that can be expressed in terms of the error function

$$\epsilon_{\bigcirc'}(\mathbf{a}_c, \mathbf{b}_c) = \epsilon_{\mathrm{SM}}(\epsilon_\Omega[\epsilon_R(\epsilon_{\mathbf{a}}), \epsilon_R(\epsilon_{\mathbf{b}})]), \tag{326}$$

where we assume that ϵ_R is known a priori, and ϵ_{SM} was given previously. This result follows straightforwardly from the composition of functions implied in Theorem 9.3.2.1.

9.3.2.4. Remark. The information loss incurred in processing transform-coded imagery can be problematic when cascading compressive operations. However, with error reporting and control incorporated into a given algorithm, one could predict which transform coefficients are more likely to be perturbed. For example, if one simulates a low-pass filter over Fourier transform-coded imagery, the high-frequency coefficients would be rejected, but the mid-range portion of the spectrum would be moderately attenuated. Computational error is associated with this attenuation, particularly as one approaches the high frequencies, where coefficient magnitudes are small and SNR is more noise- or error-sensitive. Such computational errors can induce significant error in the inverse Fourier transform, yielding mid- to high-frequency artifacts in the decompressed image.

An instance of transform coding that is currently in vogue is JPEG compression, which is discussed in general form as follows.

9.4. JPEG.

JPEG is an image compression transform that exploits spatial properties of source imagery to yield moderate compression ratios (typically, 15:1 to 30:1 in images of natural scenes). JPEG has many variants, each of which generally has proprietary parameters or processes. Thus, for purposes of simplicity, we present JPEG, which has many variants, in terms of a general formulation, which is a composition of three well-known transforms: (1) the Discrete Cosine Transform, (2) the sparse-matrix reduction transform, and (3) the Huffman encoding transform. The former two transforms are lossy, whereas Huffman transformation is lossless.

9.4.1. Background.

Due to the requirements of a finite-state automaton (FSA) and training-set specific codebook for decoding Huffman-encoded imagery, we prefer to discuss JPEG in terms of its Huffman-decoded form. We thus avoid issues of Huffman codebook construction that have been discussed elsewhere. We begin by deriving operations that process over the sparse-matrix reduction of discrete cosine transform coefficients.

9.4.1.1. Definition. The discrete cosine transform expresses an image $\mathbf{a} \in \mathbf{R}^{\mathbf{Z}_n}$ as a weighted sum of cosine functions that is given by

$$[T_{\mathrm{DCT}}(\mathbf{a})](u) = \frac{2 \cdot c(u)}{n} \sum_{x=0}^{n-1} \mathbf{a}(x) \cdot \cos\left(\frac{(2x+1)u\pi}{2n}\right) \qquad u = 0, 1, \ldots, n-1, \tag{327}$$

where

$$c(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u = 1, 2, \ldots, n-1 \end{cases}. \tag{328}$$

The *inverse 1-D discrete cosine transform* $T_{\mathrm{DCT}}^{-1}(\mathbf{a})$ is given by

$$[T_{\mathrm{DCT}}^{-1}(\mathbf{a})](x) = \sum_{u=0}^{n-1} c(u) \cdot \mathbf{a}(u) \cdot \cos\left(\frac{(2x+1)u\pi}{2n}\right). \tag{329}$$

Given a discrete image $\mathbf{a} \in \mathbf{R}^{\mathbf{Z}_n \times \mathbf{Z}_n}$, the two-dimensional cosine transform and its inverse are given by

$$\left[T_{\mathrm{DCT}}^{-1}(\mathbf{a})\right](u,v) = \frac{2 \cdot c(u) \cdot c(v)}{n} \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} \mathbf{a}(x,y) \cdot \cos\left(\frac{(2x+1)u\pi}{2n}\right) \cdot \cos\left(\frac{(2y+1)v\pi}{2n}\right) \quad (330)$$

and

$$\left[T_{\mathrm{DCT}}^{-1}(\mathbf{a})\right](x,y) = \frac{2}{n} \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} c(u) \cdot c(v) \cdot \mathbf{a}(u,v) \cdot \cos\left(\frac{(2x+1)u\pi}{2n}\right) \cdot \cos\left(\frac{(2y+1)v\pi}{2n}\right), \quad (331)$$

where c was defined previously.

We say that T_{DCT} is a *block DCT* when the transform is applied to individual encoding blocks in a source image.

9.4.1.2. Complexity and computational error. Over an M×N-pixel array $\mathbf{X}$, the separable fast formulation of the block DCT [85] applied to a k×l-pixel image has complexity given by

$$W_{\mathrm{DCT}}(M,N,k,l) = \left\lceil \frac{M}{k} \right\rceil \cdot \left\lceil \frac{N}{l} \right\rceil \cdot (l \cdot \log k + k \cdot \log l) \text{ real multiplications.} \quad (332)$$

Similar to the separable Fourier transform, the separable block DCT is comprised of $\lceil \log k \rceil + \lceil \log l \rceil$ stages where the principal operations are real addition and multiplication. Since the dataflow graph of a function f is isomorphic to that of its error function ϵ_f [13], the computational error in the DCT is given by:

$$\epsilon_{\mathrm{DCT}}(\epsilon_{\mathbf{a}}) = \bigcirc_{i=1}^{\lceil \log k \rceil + \lceil \log l \rceil} 2 \cdot \epsilon_{\times}(i) \quad \text{where} \quad \epsilon_{\times}(i) \equiv \begin{cases} \epsilon_{\times}(\epsilon_{\mathbf{a}}, \epsilon_{\mathbf{a}}) & \text{if } i = 1 \\ \epsilon_{\times} & \text{otherwise} \end{cases} \quad (333)$$

and $\epsilon_{\mathbf{a}}$ denotes the error function of the source image. Here, multiplication of $\epsilon_{\times}$ by two implements the additive error propagation law. Although we assume (for purposes of simplicity) that the error function of each block resembles $\epsilon_{\mathbf{a}}$, such ergodicity generally does not occur in imagery that is obtained from a realistic sensor whose noise figure is spatially variant. In such cases, one must calculate ϵ_{DCT} at each location in each block of the source image, which is prohibitive for production processing. Thus, we are currently investigating

the design and implementation of error protocols that would characterize error throughout an image, or in partitions (e.g., quadrants) of an image. Results will be reported in a future publication.

9.4.1.3. Definition. The *generalized JPEG transformation* that we employ in this study is the composition of the discrete cosine transform, the sparse matrix reduction transform, and the Huffman transform. Given an image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$,

$$T_{\mathrm{JPEG}}(\mathbf{a}) = [T_{\mathrm{DCT}} \circ T_{\mathrm{SM}} \circ T_{\mathrm{H}}](\mathbf{a}), \tag{334}$$

with associated parameters such as block dimensions k,l and negligible values in $S \subset \mathbf{R}$ understood.

9.4.1.4. Observation. Huffman encoding, the third component of JPEG, is a lossless transform that is codebook-based. Similar to VQ, the Huffman codebook must be computed for each training set and is thus data-dependent. Additionally, it is difficult to process over the range space of the Huffman transform $T_{\mathrm{H}}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{W}^{\mathbf{Y}} \times (\mathbf{F}^{\mathbf{m}})^{\mathbf{W}}$, due to (a) variable blocksize m and (b) the requirement of a finite-state automaton for decoding the unique prefix codes that comprise exemplars in the Huffman codebook. As a result, we assume that the Huffman-encoded data is Huffman-decoded prior to JPEG processing. Note that proprietary Huffman codebooks that are a modification of the standard Huffman codebook are employed in much commercial JPEG compression software and hardware. Thus, the development of compressive operations over the formats produced by such transforms is a difficult goal, since the Huffman encoding is obscure. Thus, we have further justification for using the generalized JPEG transform.

9.4.1.5. Complexity and information loss. The JPEG transform exhibits computational cost that is the sum of the work requirement of the DCT, sparse-matrix reduction, and Huffman transforms. Since we are concerned primarily with the former two transforms, we write W_{JPEG}, the work required by the separable fast DCT version of T_{JPEG}, as

$$\begin{aligned} W_{JPEG}(|\mathbf{X}|, k, l) &= W_{DCT}(|\mathbf{X}|, k, l) + W_{SM}(|\mathbf{X}|) \\ &= \left\lceil \frac{|p_1(\mathbf{X})|}{k} \right\rceil \cdot \left\lceil \frac{|p_2(\mathbf{X})|}{l} \right\rceil \cdot kl(\log k + \log l) \text{ multiplications and cosines} \\ &\quad + 2 \cdot |\mathbf{X}| \text{ comparisons} \end{aligned} \tag{335}$$

which can be rendered more efficient implementationally by encoding the cosine functions in a lookup table.

Assume that a source image $\mathbf{a} \in F^{\mathbf{X}}$ and that the set of negligible values S is a subset of $range(choice(range(T_{JPEG}))) = range(\mathbf{a})$. From Equation 334 and the introductory discussion, we express ϵ_{JPEG}, the error associated with JPEG, as:

$$\epsilon_{JPEG}(\mathbf{a}) = \epsilon_{SM}(S, T_{DCT}(\mathbf{a}), \epsilon_{DCT}(\epsilon_{\mathbf{a}})), \tag{336}$$

where ϵ_{SM} was given in Equation 316 and ϵ_{DCT} was given in Equation 333.

9.4.2. Pointwise Arithmetic.

We next summarize JPEG-based pointwise operations.

9.4.2.1. Theorem. An analogous operation $\bigcirc'$ over JPEG-compressed imagery is an $\bigcirc$-property of the DCT, provided that $\bigcirc'$ is refined to accomodate data structures that encode the reduced matrix of DCT coefficients.

Proof. Let the givens hold and let the JPEG transform be as stated in Equation 334, i.e., $T_{JPEG}(\mathbf{a}) = T_{SM}(T_{DCT}(\mathbf{a}))$. From the definitions of an analogue and the TPP, if $\bigcirc'$ is an $\bigcirc$-property of the DCT, then $\bigcirc'$ is an analogue of $\bigcirc$ over $range(T_{DCT})$. From Theorem 9.2.2.1, if $\bigcirc'$ is refined to accomodate T_{SM}'s data structures, then theorem holds.

□

9.4.2.2. Complexity and information loss. Given the preceding theorem, the complexity of the analogous JPEG operation is merely the complexity of the analogous DCT operation $\bigcirc'$ together with the overhead required for manipulation of reduced-matrix data structures.

Similarly, given source images $\mathbf{a}, \mathbf{b} \in \mathbf{R}^{\mathbf{X}}$ and compressed images $\mathbf{a}_c = T_{\mathrm{JPEG}}(\mathbf{a})$ and $\mathbf{b}_c = T_{\mathrm{JPEG}}(\mathbf{b})$, with $\epsilon_{\bigcirc'}$ denoting the error function of the analogous DCT operation $\bigcirc'$, then the total error in the resulting JPEG operation is given by

$$\epsilon_{\bigcirc'_{\mathrm{JPEG}}} \leq \epsilon_{\mathrm{SM}}(\epsilon_{\bigcirc'}[\epsilon_{\mathrm{DCT}}(\epsilon_{\mathbf{a}}), \epsilon_{\mathrm{DCT}}(\epsilon_{\mathbf{b}})]), \tag{337}$$

where $\epsilon_{\mathbf{a}}$ and $\epsilon_{\mathbf{b}}$ denote source image errors, with ϵ_{SM} and ϵ_{DCT} defined previously. The practical formulation of $\epsilon_{\bigcirc'}$ depends upon parameters of the JPEG implementation (e.g., quality, smoothing, sampling area). For example, if the quality setting (whose value varies with the implementation and does not indicate percent information retained) is set at a high level, then the compression ratio and representational error will be reduced. Similarly, if the smoothing parameter (appropriate for monospectral images) is set high, then local averaging tends to degrade much high-frequency data.

9.4.3. Pixel-Level Operations over JPEG-compressed Imagery.

9.4.3.1. Observation. The JPEG quantizer may be implemented either as a division process [7] or as value selection by sparse-matrix reduction. Since low frequency coefficients generally exhibit greater magnitude than high-frequency coefficients in cosine transformations of natural scenes, it is reasonable to assume that low-order coefficients of the DCT are more likely to be preserved. Thus, it is reasonable to expect that image smoothing, which can also be portrayed as low-frequency filtering, can be accomplished by filtering the output of the JPEG transform to reject all high frequency information (i.e., low-pass filtering).

9.4.3:2. Theorem. Let Observation 9.4.3.1 hold, and assume that $\mathbf{a}_c = T_{\mathrm{JPEG}}(\mathbf{a})$ has high-frequency coefficients stored as pixel values $\mathbf{a}(\mathbf{x})$. If the domain points $\mathbf{x}$ at which this information is located in the DCT transform representation are collected in set D, then the JPEG filtering algorithm that corresponds to the image operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ has an analogue $\bigcirc': (\mathbf{X}\times\mathbf{F})^{\mathbf{Y}} \to (\mathbf{X}\times\mathbf{F})^{\mathbf{Y}}$ which is denoted by:

$$\bigcirc'(\mathbf{a}_c) = \mathbf{a}_c\|_{\{(\mathbf{x},\mathbf{a}(\mathbf{x})):\,\mathbf{x}\in D\}}. \tag{338}$$

Proof. Assuming that the givens hold, D represents the domain selection criteria for high-frequency filtering. Equation 338 implements the restriction of $\mathbf{a}_c$ to domain points in D. Thus, the theorem holds.

□

9.4.3.3. Complexity. Assume, for purposes of realism, that smoothing is computed via convolution of a source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ with a template $\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ using an image operation $\bigcirc: \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$. If the compressive analogue $\bigcirc': (\mathbf{X}\times\mathbf{F})^{\mathbf{Y}} \to (\mathbf{X}\times\mathbf{F})^{\mathbf{Y}}$ over *range*(T_{JPEG}) is given by Equation 338, then the preceding theorem gives rise to a CCS

$$X_{\bigcirc} = (\mathcal{A},\zeta,\mathcal{B}) = \left(\left\{\mathbf{F}^{\mathbf{X}},(\mathbf{F}^{\mathbf{X}})^{\mathbf{X}},\bigcirc\right\},\{T,\,U;\Theta\},\left\{(\mathbf{X}\times\mathbf{F})^{\mathbf{Y}},(\mathbf{F}^{\mathbf{X}})^{\mathbf{X}},\bigcirc'\right\}\right), \tag{339}$$

where $\bigcirc' = \Theta(\bigcirc)$ and U is the null transform. That is, the template used in the image domain is present in the compressed domain merely as a placeholder, such that the elements of algebras $\mathcal{A}$ and $\mathcal{B}$ exhibit a one-to-one correspondence in number and type.

Given that $\mathbf{X}$ is m-dimensional, Equation 338 requires work of at most $m\cdot|\mathbf{Y}|$ comparison operations and $(m-1)\cdot|\mathbf{Y}|$ combinations of the comparison results (e.g., logical *and* operations). If $m < \mathrm{CR_d}$, then the computational efficiency associated with X is given by

$$\eta(X) = \frac{W_{\bigcirc}(|\mathbf{X}|)}{W_{\bigcirc'}(|\mathbf{Y}|)} \approx \frac{|\mathbf{X}|\cdot\mathcal{S}(\mathbf{t}_{\mathbf{x}})}{2m\cdot|\mathbf{Y}|} = \frac{\mathcal{S}(\mathbf{t}_{\mathbf{x}})}{2m}\cdot \mathrm{CR_d},\ \mathbf{x}\in\mathbf{X}. \tag{340}$$

For example, if $\mathbf{t}$ is a space-invariant template such as the von Neumann template, then $|\mathcal{S}(\mathbf{t}_{\mathbf{x}})| = 5$. If the encoding blocks are four pixels square, then $m = 2$ and $\mathrm{CR_D} = 16$. Under

the artificial assumption that image-template operations of multiplication and addition constituent to $\bigcirc$ require work equal to that of the comparison operations present in $\bigcirc'$, we have $\eta = (5/4)16 = 1.25(16) = 20$. However, the image-template operations are usually more burdensome than the comparisons, so $\eta > 20$ would be more likely in this case.

9.4.3.4. Information loss. The method of Equation 338 can be analyzed similarly, since values of pixels in $p_2(\mathbf{a}_c(\mathbf{Y}))$ that correspond to preselected domain points in $\mathbf{X}$ are discarded during filtering. Denoting the domain points of discarded pixels as $\mathrm{B} = p_1(\mathbf{a}_c(\mathbf{Y}))$ \ D, the information discarded by JPEG filtering is given by

$$\overline{\epsilon_{\mathrm{JPf}}}(\mathrm{D}, \mathbf{a}_c) = \frac{1}{|\mathbf{X}|} \cdot \sum_{\mathbf{x} \in \mathrm{B}} p_2(\mathbf{a}_c(\mathbf{x})) . \tag{341}$$

9.4.3.5. Example. Given an image database of outdoor scenes containing small man-made targets that were acquired using an intensified camera with approximately three bits per pixel error, we computed the error and performance measures shown in Figure 15 for pointwise image addition. This operation, although simple mathematically, was useful for averaging images over a multispectral frame containing six monospectral images. The abscissas of Figure 15 depict removal of the lower first through fourth magnitude quintiles of the DCT coefficients in the matrix reduction process. The zero-th quintile merely represents the control case, i.e., where all transform coefficients were preserved. Error is expressed on the left-hand ordinate in terms of bits per pixel, and computational efficiency is expressed on the right-hand ordinate in terms of the factor by which the speed of noncompressive addition was increased. Note that for the zero-th quintile, the efficiency is slightly less than unity, due to the overhead incurred by accessing reduced-matrix data structures. This overhead is further reflected in the fourth-quintile efficiency, which is less than the theoretical maximum of $5 = 1/(1\text{–}0.8)$.

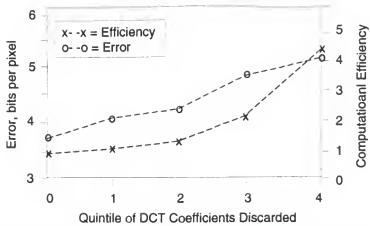


Figure 15. Error and efficiency measures associated with JPEG addition over natural scenes having 3 bits/pixel error.

We consider this example to describe a limiting case of computational error, since the imagery is already severely noise-degraded, and the error upper bound is described by additive error propagation. We believe that the variance from linearity in the error result is due to two factors. First, the DCT coefficients that were discarded represent much high-frequency information, such as texture of grass, which was prevalent as background. Additionally, there were patches of sand among the grass that contributed high- and mid-range frequency information, which was obliterated as the third and fourth quintiles were neglected. Second, various multispectral bands of the imagery had different modulation transfer functions (MTFs). Thus, the DCT coefficients that were discarded perturbed the representational error of different monospectral band images in different ways. This result has important implications for the accuracy of compressive ATR filters, and will be investigated in future research.

9.4.3.6. Observation. JPEG-based filtering in the spatial frequency domain was described in Equation 338. Thus, JPEG-based edge detection, which can be implemented as high-pass filtering, would be computed using Equation 338, where D would contain the domain points of high-frequency coefficients of the DCT.

9.5. Block Truncation Coding and VPIC.

We begin by reviewing the BTC and VPIC transforms, then derive operations over BTC- and VPIC-compressed imagery.

9.5.1. Basic Concepts of BTC.

Block truncation coding [16] stores the mean of an encoding block. If the block standard deviation exceeds a prespecified threshold, an *edge block* is said to be present, and the standard deviation is stored, together with a bitmap of the block crossings about the mean.

9.5.1.1. Algorithm. If Definitions 9.1.1.1 and 9.1.1.2 hold, then for each source image block $\mathbf{b}(\mathbf{y})$, $\mathbf{y} \in \mathbf{Y}$, the BTC transform is computed as follows:

Step 1. Determine the block mean $\mu(\mathbf{y})$ and standard deviation $\sigma(\mathbf{y})$.

Step 2. Assuming that the compressed image $\mathbf{a}_c$ is initialized to some constant f, where $f \ll \wedge range(\mathbf{a})$ or $f \gg \vee range(\mathbf{a})$, and given variance threshold T_{min}, interrogate the standard deviation as follows:

$$\mathbf{a}_c(\mathbf{y}) = \mu \quad \text{if} \quad \sigma(\mathbf{y}) < T_{min}. \tag{342}$$

If $\mathbf{a}_c(\mathbf{y}) = f$, then an *edge block* is present, and we proceed to Step 3. Otherwise, a mean block is present, and we apply Step 1 to the $\mathbf{y}+1$-th block.

Step 3. Compute the bitmap $\mathbf{d}$ of the zero crossings of $\mathbf{b}(\mathbf{y})$ about its mean, as follows:

$$\mathbf{d} = \chi_{>0}(\mathbf{b}(\mathbf{y}) - \mu). \tag{343}$$

Step 4. Via bitmap manipulations that vary with the implementation of the BTC transform, reduce the entropy of $\mathbf{d}$ by eliminating single bits in rows or columns,

to yield clusters of four or more bits if $k,l > 2$. Such reductions can be performed in several ways. For example, given a bit pattern such as

$$\mathbf{d} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \tag{344}$$

the leftmost group of two unitary bits, as well as the single unitary bit in the bottom row, would be zeroed to form the manipulated bitmaps

$$\hat{\mathbf{d}}_1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{or} \quad \hat{\mathbf{d}}_2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{345}$$

Note that the zeroes present in the left-hand half of $\mathbf{d}$ have been interchanged with neighboring unitary values to produce $\hat{\mathbf{d}}_1$, which can be thought of as being comprised of two 2×2-pixel sub-blocks of form $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. In contrast, $\hat{\mathbf{d}}_2$, is comprised of two overlapping unitarily-valued 2×2-pixel clusters. Such construction can facilitate more efficient bitmap coding in terms of preselected exemplar patterns or hierarchical indexing.

Step 5. Store the following parameters in the compressed representation of the $\mathbf{y}$-th edge block

$$\mathbf{a}_c(\mathbf{y}) = (\mu, \sigma, \mathbf{d}). \tag{346}$$

Subsequently, encoded blocks are binarized to effect maximum bit packing. For example, the mean value may be quantized to 5 bits and the standard deviation to 3 bits.

Decompression involves the block reconstruction step

$$\tilde{\mathbf{b}} = \sigma \cdot (2\mathbf{d} - 1) + \mu, \tag{347}$$

which requires one multiplication and $2kl$ additions per edge block as well as kl I/O operations per block.

9.5.1.2. Complexity. Per $k \times l$-pixel block, BTC requires one comparison operation to threshold the standard deviation, $kl - 1$ additions and one multiplication to compute the block mean, and $2kl$ subtractions and multiplications, as well as one square root to compute the standard deviation. Per block, one I/O operation is required for storing the mean value. Per edge block, one I/O operation is employed to store the standard deviation and at least $\mathbf{O}(kl)$ comparison operations are required for block manipulation. Neglecting I/O overhead and letting $|\mathbf{Y}| = \lceil M/k \rceil \cdot \lceil N/l \rceil$ denote the number of encoding blocks, the total computational cost of BTC is bounded as

$$\begin{aligned} W_{BTC}(|\mathbf{Y}|, k, l) \leq\ & |\mathbf{Y}| \cdot (3kl - 1 \text{ additions } + 2kl \text{ multiplications} \\ & (+\,1 \text{ sqrt } + \mathbf{O}(kl) \text{ comparisons}) \,, \end{aligned} \tag{348}$$

where *sqrt* denotes the square root operation.

9.5.1.3. Information loss. Due to (a) the representation of a "smooth" block by the block mean, as well as (b) the representation of an edge block by the block mean, standard deviation σ, and manipulated bitmap $\mathbf{d}$, decompression of the BTC transform output approximates the source image. Observe that BTC is designed to represent a source image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ in terms of a reconstructed image $\tilde{\mathbf{a}} \in \mathbf{R}^{\mathbf{X}}$, within a prespecified error limit $\epsilon \propto \|\tilde{\mathbf{a}} - \mathbf{a}\|$ that can be expressed in terms of the block standard deviation σ under the assumption of normality.

A more convenient method is to let the location of the i-th point within the $\mathbf{y}$-th encoding block be given by $\mathbf{x} = h^*(h(\mathbf{y}))(i) \in \mathbf{X}$. The probability of a value $\mathbf{e}(\mathbf{x})$ in the error image $\mathbf{e} = \tilde{\mathbf{a}} - \mathbf{a}$ given the corresponding source value $\mathbf{a}(\mathbf{x})$ is given by

$$\Pr(\mathbf{e}(\mathbf{x})) = \begin{cases} 0.683 & \text{if } \mathbf{a}(\mathbf{x}) \in [\mu - \sigma, \mu + \sigma] \\ \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-(\mathbf{a}(\mathbf{x}) - \mu)/2\sigma^2} & \text{otherwise} \end{cases}, \tag{349}$$

where the value 0.683 is the probability of error at one standard deviation. Since the probability distribution of source values in a reconstructed block is distorted by quantizing block values to the interval $[\mu - \sigma, \mu + \sigma]$, nothing more can be said about the associated error. Given μ and σ, similar expressions can be constructed from non-normal distributions.

9.5.2. Visual Pattern Image Coding.

We next describe visual pattern image coding, which resembles BTC since block mean and a representation of block variance are included in the compressed image. Bitmap manipulation is not a feature of VPIC, which matches a zero-crossing bitmap to a small number of codebook exemplars to achieve a best-match exemplar whose index is stored in the compressed image, versus simplification and storage of the zero-crossing bitmap employed in BTC.

9.5.2.1. Algorithm. If Definitions 9.1.1.1 and 9.1.1.2 hold, then for each block $\mathbf{b}(\mathbf{y})$, $\mathbf{y} \in \mathbf{Y}$, the VPIC transform is computed as follows:

Step 1. Determine the block mean $\mu(\mathbf{y})$ and partial gradients $\triangle_x$, $\triangle_y$ as follows

$$\mu = \frac{\Sigma \mathbf{b}(\mathbf{y})}{|h^*(\mathbf{y})|}, \quad \triangle_x = \frac{\mathbf{b}_t}{|domain(\mathbf{b}_t)|} - \frac{\mathbf{b}_b}{|domain(\mathbf{b}_b)|}, \text{ and}$$
$$\triangle_y = \frac{\mathbf{b}_l}{|domain(\mathbf{b}_l)|} - \frac{\mathbf{b}_r}{|domain(\mathbf{b}_r)|}, \tag{350}$$

where $\mathbf{b}_t$, $\mathbf{b}_b$, $\mathbf{b}_l$, and $\mathbf{b}_r$ denote the top, bottom, left, and right halves of $\mathbf{b}(\mathbf{y})$. If $\mathbf{b}$ has an odd number of rows, then the first $\lceil l/2 \rceil$ pixels of $\mathbf{b}$'s middle row are contained in $\mathbf{b}_t$, and symmetrically for the k columns of $\mathbf{b}_l$. The gradient intensity $g(\mathbf{y})$ and orientation $\theta(\mathbf{y})$ are given by

$$g(\mathbf{y}) = \sqrt{\triangle_x^2 + \triangle_y^2} \quad \text{and} \quad \theta(\mathbf{y}) = \tan^{-1}(\triangle_y / \triangle_x). \tag{351}$$

Step 2. Given that the compressed image $\mathbf{a}_c$ is initialized to a constant value f, where $f \ll \wedge range(\mathbf{a})$ or $f \gg \vee range(\mathbf{a})$, and given threshold T_{min} and the saturation threshold T_{max}, we interrogate the gradient intensity as follows:

$$\mathbf{a}_c(\mathbf{y}) = \begin{cases} \mu & \text{if } g(\mathbf{y}) < T_{min} \\ T_{max} & \text{if } g(\mathbf{y}) \geq T_{max} \end{cases}. \tag{352}$$

If $\mathbf{a}_c(\mathbf{y}) = f$, then an edge block is present and we proceed to Step 3. Otherwise, a mean block is present and processing of $\mathbf{b}(\mathbf{y})$ terminates in favor of applying Step 1 to the $\mathbf{y}+1$–th block.

Step 3. Compute the bitmap $\mathbf{d}$ of the zero crossings of $\mathbf{b}(\mathbf{y})$ about its mean, as

$$\mathbf{d} = [2 \cdot \chi_{>0}(\mathbf{b}(\mathbf{y}) - \mu)] - 1 , \tag{353}$$

which can be binarized by substituting 0 for −1.

Step 4. Let $\mathbf{d}$ be correlated with bitmap exemplars in the codebook $\mathbf{c} \in \left(\{0,1\}^{kl}\right)^{\mathbf{U}}$, where finite $\mathbf{U} \subset \mathbb{N}$. Implementationally, we assume that correlation is assisted by a LUT which restricts the choice of exemplars in $\mathbf{c}$ to orientations near θ. For example, the bitmap $\mathbf{d}$ portrayed in Equation 344 could be matched to an exemplar such as

$$\mathbf{c}(\mathbf{u}) = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \mathbf{u} \in \mathbf{U} . \tag{354}$$

The exemplar index $\mathbf{u}$ is stored with the edge block parameters in the compressed image as

$$\mathbf{a}_c(\mathbf{y}) = (\mu, g, \theta, \mathbf{u}) . \tag{355}$$

After all blocks are encoded, $\mathbf{a}_c$ is binarized to achieve maximum bit packing.

Decompression involves scaling the $\mathbf{u}$-th exemplar pattern for mean μ and gradient g, where the latter can be thought of as being similar to the standard deviation in Equation 347. The decompression step therefore requires one multiplication and kl additions per edge block, as well as kl I/O operations per mean block.

9.5.2.2. Observation. Horizontal and vertical block partitions (Step 1) are recommended [9], which is likely due to the implications of psychophysical research pertaining to the vertical and horizontal primary and secondary directions of gaze [74,75]. We are currently investigating additional methods of block partitioning that may facilitate optimization of VPIC's compression ratio. For example, overlapping elliptical or hexagonal blocks would respectively be appropriate as a consequence of overlapping visual fields or close-packing of

retinal photoreceptors. Additionally, we note that the preceding algorithm can be computed in SIMD-parallel fashion, since μ, g, θ, and $\mathbf{u}$ can be stored as images on $\mathbf{Y}$. When $\mu < T_{min}$, the unused values (g, θ, and $\mathbf{u}$) in Equation 355 would be ignored in the binarization step, which could be computed in SIMD-parallel fashion, followed by a merging step that would optimally pack $\mathbf{a}_c$. This issue is further discussed in Chapter 11.

9.5.2.3. Remark. Our preliminary results indicate that following VPIC with Huffman encoding may yield near-optimal compression. However, the Huffman codebook must be recomputed for each VPIC encoding of an image training set, due to the non-ergodic distribution of mean and standard deviation values in various test sets. For example, training imagery may have Gaussian-distributed means, while test-set imagery acquired using a different sensor may have block means may conform to a Poisson distribution. Recall that it is not convenient to process over Huffman-encoded imagery, due to nonuniform blocksize and the implementational requirement of a finite-state automaton (FSA) in the decoding step. In Section 3.4.8, we showed that nonuniform blocksize could induce block fragmentation in compressive binary pointwise and image-template operations. FSA-based decoding is not necessarily useful for other types of compression formats, and would represent additional complexity useful only for Huffman decoding. If Huffman encoding follows VPIC, then it is reasonable in this study to assume Huffman decoding prior to processing VPIC–encoded imagery.

9.5.2.4. Complexity. Chen and Bovik [9] showed that VPIC requires the following operations:

- *Per block*, kl–1 additions and one multiplication for each computation of the block mean, and one subtraction and two comparisons to compute the gradient intensity g, since (a) $\Sigma\mathbf{b}_t$ and $\Sigma\mathbf{b}_b$ can be added to obtain $\Sigma\mathbf{b}$ for computing the mean and (b) division is not required to compute non-normalized g for purposes of comparison with T_{min}.

- *Per edge block*, VPIC requires $4k\ell+2$ additions, $k\ell$ multiplications, one square root and one inverse tangent operation to compute the gradient orientation θ. Depending on the block matching scheme, if the codebook has $|\mathbf{U}|$ exemplars, the matching overhead is bounded above by $2k\ell \cdot |\mathbf{U}|$ subtractions. Additionally, $2 \cdot |\mathbf{U}|$ multiplications and $|\mathbf{U}|$ square root operations are required for rms computation of matching scores.

Letting $|\mathbf{Y}| = \lceil M/k \rceil \cdot \lceil N/\ell \rceil$ denote the number of encoding blocks and N_e denote the number of edge blocks, the total computational cost is bounded as

$$\begin{aligned} W_{VPIC}(|\mathbf{X}|, k, \ell, |\mathbf{U}|) \leq\ & |\mathbf{X}| \cdot \left(5 + \frac{1}{k\ell}\right) + |\mathbf{U}| \cdot 2k\ell \ \text{ additions} \\ & + |\mathbf{X}| \cdot \left(1 + \frac{1}{k\ell}\right) + |\mathbf{U}| \ \text{ multiplications} \\ & + |\mathbf{X}|/k\ell + |\mathbf{U}| \ \text{ roots } + |\mathbf{X}|/k\ell \ \text{inverse tangents}. \end{aligned} \tag{356}$$

If $|\mathbf{U}| \ll |\mathbf{X}|/k\ell$, which is customary, and if $k\ell$ is sufficiently large, then W_{VPIC} can be approximated in practice by six additions and one multiplication per pixel of the source image, which is a small cost. For example, if $|\mathbf{X}| = 1M$, then 6M additions and 1M multiplications would be required.

9.5.2.5. Information loss. As in the case of BTC, due to (a) the representation of a "smooth" block by the block mean, as well as (b) the representation of an edge block by the block mean, gradient intensity g, orientation θ, and exemplar index $\mathbf{u}$, VPIC approximately represents the source image and is therefore lossy. In this study, we assume that the gradient g approximates the standard deviation, since both are computed from block means in RMS fashion. As a result, the error of estimation under which Equation 349 applies depends primarily upon (a) the codebook exemplars, (b) codebook size constraints, and (c) the frequency of exemplar use, the latter of which depends upon encoding block configurations in the training set.

9.5.2.6. Remark. We further note that VPIC error analysis may be considered a moot point for image display. This conclusion is supported by the fact that VPIC is designed to produce a visually pleasing (versus a mathematically accurate) representation of the source image. As previously mentioned, this effect results from a prespecified codebook whose exemplars are designed to conform to knowledge about patterns that stimulate cortical receptive fields [74,75]. As a result, VPIC is not designed to minimize a criterion such as mean-squared error (MSE), which is the case for most compression transforms (e.g., BTC, JPEG, or VQ). Thus, VPIC appears to be more appropriate for visual analysis of image content, versus ATR applications that emphasize computerized image analysis under uniform error constraints.

9.5.3. Pointwise Arithmetic.

We have found that operations of addition, multiplication, and maximum can be implemented over the range spaces of BTC- and VPIC-encoded imagery. VQ-based arithmetic is more involved, due to issues of codebook growth and information loss. Salient discussion follows.

9.5.3.1. Assumption. Let a source image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ by compressed by a BTC or VPIC transform $T\colon \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{G}^{\mathbf{Y}}$ to yield $\mathbf{a}_c \in \mathbf{G}^{\mathbf{Y}}$, then be decompressed to yield a reconstructed image $\tilde{\mathbf{a}} \in \mathbf{R}^{\mathbf{X}}$. Let the bipolar transform $b(\mathrm{f}) = 2\mathrm{f} - 1$, $\mathrm{f} \in \{0,1\}$. Assume that a pointwise image operation $\bigcirc\colon \mathbf{R}^{\mathbf{X}} \times \mathbf{R}^{\mathbf{X}} \rightarrow \mathbf{R}^{\mathbf{X}}$ has an analogue $\bigcirc'$ over *range*(T), which comprises a CCS $X_{\bigcirc} = (\{\mathbf{R}^{\mathbf{X}}, \bigcirc\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \bigcirc'\})$, where $\bigcirc' = \Theta(\bigcirc)$.

9.5.3.2. Theorem. If Assumption 9.5.3.1 holds, T = BTC, $\bigcirc = +$, $\mathrm{g} = \mathbf{a}_c(\mathbf{y})$, and $\mathrm{h} = \mathbf{a}_c(\mathbf{z})$, where $\mathbf{y},\mathbf{z} \in \mathbf{Y}$, then $\bigcirc'$ is given by:

$$\bigcirc'(\mathrm{g},\mathrm{h}) = \begin{cases} \sum\limits_{\mathrm{f} \in \{\mathrm{g},\mathrm{h}\}} p_1(\mathrm{f}) + p_2(\mathrm{f}) \cdot b(p_3(\mathrm{f})) & \text{if g and h represent edge blocks} \\ p_1(\mathrm{g}) + p_1(\mathrm{h}) & \text{otherwise} \end{cases}, \quad (357)$$

where p_i denotes projection to the i-th coordinate, p_1 denotes block mean μ, p_2 denotes block standard deviation σ, and p_3 denotes the block bitmap.

Similarly, if $\bigcirc = \cdot$, then

$$\bigcirc'(g,h) = \begin{cases} \prod_{f \in \{g,h\}} p_1(f) + p_2(f) \cdot b(p_3(f)) & \text{if g and h represent edge blocks} \\ p_1(g) \cdot p_1(h) & \text{otherwise} \end{cases}, \quad (358)$$

and if $\bigcirc = \vee$, then

$$\bigcirc'(g,h) = \begin{cases} \bigvee_{f \in \{g,h\}} p_1(f) + p_2(f) \cdot b(p_3(f)) & \text{if g and h represent edge blocks} \\ p_1(g) \vee p_1(h) & \text{otherwise} \end{cases}. \quad (359)$$

The operation of pointwise minimum is symmetrically defined.

Proof. The proof is outlined as follows. Assuming that the givens hold, denote the arguments of $\bigcirc'$ as g and h. Equation 357 merely denotes the summation of reconstructed blocks corresponding to $\mathbf{a}_c(\mathbf{y})$, and $h = \mathbf{a}_c(\mathbf{z})$, where $\mathbf{y},\mathbf{z} \in \mathbf{Y}$. The proof is symmetric for $\bigcirc = \cdot, \vee$, and the theorem holds.

□

9.5.3.3. Observation. Equation 357 can be characterized in terms of a map $f: (\mathbf{R}^2 \times \mathbf{N})^2 \rightarrow \mathbf{R}^2 \times \mathbf{N}$, where $p_2(range(f))$ denotes the index of a manipulated bitmap. Implementationally, the mean and standard deviation (or gradient, in the case of VPIC) would be indexed and collected together with the exemplar index to yield $f: \mathbf{N}^3 \times \mathbf{N}^3 \rightarrow \mathbf{N}^3$. Such implementation is appropriate for BTC, since μ and σ are quantized and the number of manipulated bitmap configurations M is small in relation to the number (2^{kl}) of possible bitmaps. In practice, we usually assume $M \leq 32$ for $k,l = 4$, since manipulation yields only those blocks that have unitarily-valued components of size 2×2-pixels or larger, such that $\log(M) = 5$ bits. Assuming that $siz(range(\mathbf{a})) = 8$ bits as well as 5 and 2 bits required for μ and σ, respectively, we have 12 bits = 5 + 5 + 2 bits required for each operand. Thus, the total LUT domain size is 24 bits, or 16 MB.

In VPIC based processing, a codebook of size M = 3 is typical for two possible angular orientations. Given 3 bits for the mean and 2 bits for the block gradient, we have 9 bits

per edge block (not including the block i.d. flag, which is not germane). This implies that *domain*(f) would have 2(9) = 18 bits. Since each value in *range*(f) would have 9 bits, the LUT implementation of f would have total size 294,912 bytes, which is feasible implementationally. This observation holds for pointwise arithmetic and logical operations.

9.5.3.4. Complexity. Equation 357 requires $3kl$ additions and $2kl$ sign manipulations per edge block, as well as one addition per mean block. Implementationally, it is reasonable to assume that the sign manipulations incur negligible cost in relationship to the additions. If a fraction r of the encoding blocks are edge blocks, then the work required by BTC pointwise addition is given by

$$W_{O'}(|\mathbf{X}|) = |\mathbf{Y}| \cdot (3r \cdot kl + r - 1) \text{ additions}. \tag{360}$$

For example, if r = 0.4 and $kl = 16$, then $18.6 \cdot |\mathbf{Y}|$ additions are required. This implies that $CR_D > 18.6$ if computational efficiency $\eta > 1$. However, this is a contradiction, since $CR_D = kl = 16$. We resolve this apparent conflict by observing that the condition $(3rkl + r - 1) < CR_D$ must be satisfied for a computational speedup to occur. We note that if $kl = 16$, then $CR_D > 49r - 1$ is required for $\eta > 1$. In contrast, if the LUT described in Section 9.5.3.3 is employed, then $r \cdot |\mathbf{Y}|$ applications of the LUT are required for each edge block, and one I/O operation is required per mean block. Neglecting I/O overhead, we have

$$\eta = \frac{W_O(|\mathbf{X}|)}{W_{O'}(|\mathbf{Y}|)} = \frac{|\mathbf{X}| \text{ additions}}{r \cdot |\mathbf{Y}| \text{ LUT accesses}} = \mathbf{O}\left(\frac{CR_D}{r}\right), \tag{361}$$

which reduces to CR_D/r, if (a) the addition operation is performed in a LUT or (b) additions and LUT accesses require equal work. If W is replaced in the preceding equation by the time complexity T, then η denotes sequential computational speedup. Since $0 < r < 1$, processing at efficiencies greater than CR_D is feasible.

9.5.3.5. Information loss. Given the operands g and h of $\bigcirc'$, if ϵ_g denotes the error in g and ϵ_h denotes the error in h, then by the additive law of error propagation, the error in $\bigcirc'$ is given by:

$$\epsilon_{\bigcirc'} = \epsilon_g + \epsilon_h \quad \text{if} \quad \bigcirc \in \{+, \vee\}. \tag{362}$$

Since we assume that the operations of maximum or minimum are implemented by subtraction or comparison, the additive error function applies. In the case of multiplication, we have the exact expression:

$$\epsilon_{\bigcirc'} = gh \cdot \left(\left| \frac{\epsilon_g}{g} \right| + \left| \frac{\epsilon_h}{h} \right| \right) + \epsilon_g \cdot \epsilon_h \quad \text{if} \quad \bigcirc = \cdot, \tag{363}$$

which is customarily approximated by the first term of the preceding outer sum, since the product of ϵ_g and ϵ_h is small.

9.5.3.6. Observation. Pointwise operations for VPIC-compressed imagery are symmetrically derived. Since VPIC lacks a direct measure of standard deviation, we prefer to think of the block gradient g as a measure of deviation. This is justified by noting that the gradient of a source encoding block $\mathbf{b}$ is given by:

$$g(\mathbf{b}) = \sqrt{\triangle_x^2 + \triangle_y^2} = \sqrt{\left(\overline{\mathbf{b}_t} - \overline{\mathbf{b}_b}\right)^2 + \left(\overline{\mathbf{b}_l} - \overline{\mathbf{b}_r}\right)^2}, \tag{364}$$

where $\mathbf{b}_t$, $\mathbf{b}_b$, $\mathbf{b}_l$, and $\mathbf{b}_r$ denote the top, bottom, left, and right halves of $\mathbf{b}$ and the overline denotes the arithmetic mean. By way of comparison, the computation of g has the same general form as the standard deviation

$$\sigma(\mathbf{b}) = \sqrt{\frac{1}{|domain(\mathbf{b})|} \cdot \sum_{\mathbf{x} \in domain(\mathbf{b})} (\mathbf{b}(\mathbf{x}) - \mu(\mathbf{b}))}, \tag{365}$$

where $\mu(\mathbf{b})$ denotes the mean of $\mathbf{b}$. The chief exception to equality between g and σ is that the hemi-block means $\overline{\mathbf{b}_t}$, $\overline{\mathbf{b}_b}$, etc., are taken in VPIC within block partitions prior to differencing.

In practice, we have found that g is often a useful estimate of σ, within an error of ± 20 percent. Since many of the operations presented herein quantize values in $\mathbf{b}$ to an interval $[\mu - n\sigma, \mu + n\sigma]$, where $1 \leq n \leq 3$ is usual, we find that the approximation of σ by g is useful in practice. This is also supported by the observation than many of the images in the database used for this study exhibit noise levels that are within ± 5 percent of full scale (fs). Assuming that σ approximates such noise, 20 percent deviation in ± 5 percent fs, which equals ± 1 percent fs, is not a significant error.

9.5.3.7. Remark. Simpler methods of computing BTC and VPIC pointwise operations occasionally suffice in high-resolution imagery when the encoding block size is small. For example, et $\bigcirc = +$ have an analogue $\bigcirc' = \bigcirc$ over $range(T_{\mathrm{VPIC}})$ such that the sum of encoding blocks $\mathbf{c} = \mathbf{a} + \mathbf{b}$. The compressed sum $\mathbf{c}_c = T_{\mathrm{VPIC}}(\mathbf{c})$ can be described in terms of block mean and standard deviation as

$$\mu_c = \mu_a + \mu_b \quad \text{and} \quad \sigma_c = \sigma_a + \sigma_b \tag{366}$$

since (a) block means combine additively by the linearity property of T_{VPIC}, and (b) deviations combine additively by the law of additive error propagation.

9.5.3.8. Complexity. The preceding method is attractive since it only requires two additions per block, for a total cost of $2 \cdot |\mathbf{Y}|$ additions. It follows that the resulting efficiency $\eta = \mathrm{CR_D}/2$, if the additional overhead incurred by block parameter I/O is neglected.

9.5.3.9. Information loss. A key problem associated with the method of Equation 366 is the indeterminacy of the zero-crossing bitmap in the sum. For high-resolution imagery we usually assume that the bitmap is merely zeroed, or has a random pattern. This obviates the $\mathbf{O}(|\mathbf{X}|)$ overhead associated with combining the encoding blocks pixelwise. If the block bitmaps are taken into account (as they should be), then a wide range of pixel sums must

be quantized into the number of bits allotted for the block mean and standard deviation. For example, letting $f_b(x) = 2x - 1$, where x is a Boolean value, the sum $\mathbf{a} + \mathbf{b}$ becomes

$$\begin{aligned}
\mathbf{a} + \mathbf{b} &\approx \mu_a + \sigma_a \cdot f_b \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} + \mu_b + \sigma_b \cdot f_b \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \\
&= \mu_a + \mu_b + \begin{bmatrix} \sigma_a & -\sigma_a & \sigma_a \\ -\sigma_a & \sigma_a & \sigma_a \\ -\sigma_a & -\sigma_a & \sigma_a \end{bmatrix} + \begin{bmatrix} -\sigma_b & \sigma_b & -\sigma_b \\ \sigma_b & -\sigma_b & \sigma_b \\ -\sigma_b & \sigma_b & \sigma_b \end{bmatrix} \qquad (367) \\
&= \mu_a + \mu_b + \begin{bmatrix} \sigma_a - \sigma_b & -(\sigma_a - \sigma_b) & \sigma_a - \sigma_b \\ -(\sigma_a - \sigma_b) & \sigma_a - \sigma_b & \sigma_a + \sigma_b \\ -(\sigma_a + \sigma_b) & -(\sigma_a - \sigma_b) & \sigma_a + \sigma_b \end{bmatrix}.
\end{aligned}$$

Thus, the block standard deviation must be quantized from the four-valued block shown in the right member of the last line of Equation 367.

9.5.3.10. Observation. The preceding problem can be remedied by quantizing the four-valued standard deviation sum to the range [p,q], where

$$p = \wedge[-(\sigma_a + \sigma_b), \sigma_a + \sigma_b, -(\sigma_a - \sigma_b), \sigma_a - \sigma_b] \qquad (368)$$

and

$$q = \vee[-(\sigma_a + \sigma_b), \sigma_a + \sigma_b, -(\sigma_a - \sigma_b), \sigma_a - \sigma_b]. \qquad (369)$$

This quantization step can be implemented in a LUT referenced by block index, which is especially useful for VPIC operations, since the codebook (and LUT) are small.

9.5.3.11. Complexity. Equation 367 requires $\mathbf{O}(|\mathbf{X}|)$ additions if implemented over BTC imagery whose zero-crossing blocks are not indexed. This contradicts the philosophy of compressive processing, since the image addition requires only $|\mathbf{X}|$ additions. However, VPIC-compressed imagery requires only $\mathbf{O}(|\mathbf{Y}|)$ LUT invocations, if Equation 367 is implemented in a LUT that accepts block mean, standard deviation and index. Such a LUT is practical for 8-bit imagery, where we usually specify the block mean, standard deviation, and index in terms of three, two, and three bits, respectively. Thus, the LUT domain size would be 16 bits, yielding a 64KB LUT, which is feasible for workstations and parallel processors with large local memories.

If I/O overhead is negligible, the efficiency resulting from Equation 367 is given by

$$\eta = \frac{W_{\bigcirc}(|\mathbf{X}|)}{W_{\bigcirc'}(|\mathbf{Y}|)} = \frac{|\mathbf{X}| \text{ additions}}{|\mathbf{Y}| + 2 \text{ additions} + |\mathbf{Y}| \text{ LUTs}} \approx \frac{|\mathbf{X}|}{|\mathbf{Y}|} = CR_d/2, \tag{370}$$

which approximates the speedup exhibited by Equation 366.

9.5.3.12. Observation. BTC- or VPIC-based pointwise multiplication is similarly derived. Given the product law of error propagation, we model the block standard deviations in the pointwise block product $\mathbf{c} = \mathbf{a} * \mathbf{b}$ as errors. Thus, we have

$$\mu_{\mathbf{c}} = \mu_{\mathbf{a}} \cdot \mu_{\mathbf{b}} \quad \text{and} \quad \sigma_{\mathbf{c}} \approx \mu_{\mathbf{c}} \cdot \left(\frac{\sigma_{\mathbf{a}}}{\mu_{\mathbf{a}}} + \frac{\sigma_{\mathbf{b}}}{\mu_{\mathbf{b}}} \right) \tag{371}$$

which is analyzed similarly to Equation 366, with the exception that three multiplications and two additions are required per block.

9.5.4. Global Reduce Operations.

We begin our discussion of global reduce operations with a simple illustration of image summation, then progress to an approximate form that can be computed in $\mathbf{O}(|\mathbf{Y}|)$ time.

9.5.4.1. Observation. The results of Theorem 9.5.3.2 can be adapted to form analogues for the global reduce operations, provided that the compressed value set $\mathbf{G}$ can portray $\Sigma\mathbf{a}$ or $\Pi\mathbf{a}$. Otherwise, only the image maximum and minimum could have analogues over *range*(T). Such operations return values in *range*($\mathbf{a}$), which is encoded in $\mathbf{G}$. In such cases, the error analysis of the preceding section holds.

In the case of image summation and product, the following theorems apply.

9.5.4.2. Assumption. Let a source image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ be compressed by the BTC transform $T\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{Y}}$ to yield $\mathbf{a}_c \in \mathbf{G}^{\mathbf{Y}}$, then be decompressed by $T^*\colon \mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{Y}}$ to yield a reconstructed image $\tilde{\mathbf{a}} \in \mathbf{R}^{\mathbf{X}}$. Assume that a global reduce operation $\Gamma\colon \mathbf{R}^{\mathbf{X}} \to \mathbf{R}$ has an analogue Γ' over *range*(T) that comprises a CCS $X_{\Gamma} = (\{\mathbf{R}^{\mathbf{X}}, \mathbf{R}; \Gamma\}, \{T, \Theta\}, \{\mathbf{G}^{\mathbf{Y}}, \mathbf{G}; \Gamma'\})$, where $\Gamma' = \Theta(\Gamma)$.

9.5.4.3. Theorem. If Assumption 9.5.4.2 holds and the following conditions hold:

(i) T = BTC,

(ii) $\mathbf{\Gamma} = \Sigma$,

(iii) $\mathbf{h} \in \mathbf{N}^{\mathbf{Z}_m}$ denotes the histogram of $\mathbf{a} \in \mathbf{Z}_m^{\mathbf{X}}$,

(iv) the compressed image $\mathbf{a}_c = T(\mathbf{a})$, and

(v) the error image $\mathbf{e} = \tilde{\mathbf{a}} - \mathbf{a}$;

then the error ϵ_Σ inherent in $\mathbf{\Gamma}'(\mathbf{a}_c) = \Sigma\, p_1(\mathbf{a}_c)$, is bounded as

$$\epsilon_\Sigma \leq \Sigma \mathbf{e}, \tag{372}$$

where p_i, i = 1..3 were defined previously. Similarly, the error ϵ_Π in the image product $\mathbf{\Gamma} = \Pi$, which has an analogue $\mathbf{\Gamma}'(\mathbf{a}_c) = \Pi\, p_1(\mathbf{a}_c)$, is bounded as

$$\epsilon_\Pi \leq \Pi \mathbf{a} \cdot \sum \left| \frac{\mathbf{e}}{\mathbf{a}} \right| + \Pi \mathbf{e}. \tag{373}$$

Proof. The proof is summarized as follows. Assuming that the givens hold, let $\mathbf{\Gamma}' = \Sigma$ be expressed as

$$\mathbf{\Gamma}'(\mathbf{a}_c) = \sum_{\mathbf{y} \in \mathbf{Y}} p_1(\mathbf{a}_c(\mathbf{y})), \tag{374}$$

which is the sum of block means μ. The error ϵ_Σ inherent in $\mathbf{\Gamma}'$ is given by the summation over all $\mathbf{y} \in \mathbf{Y}$ of the error with which $\mu(\mathbf{y})$ represents values in the y-th block. Given the error image $\mathbf{e} = \tilde{\mathbf{a}} - \mathbf{a}$, $\Sigma\mathbf{e}$ denotes the representational error in the sum (from the additive law of error propagation) since (a) the within block error due to block summation is given by $\epsilon_b(\mathbf{y}) = \Sigma\, \mathbf{e}|_{h^*(\mathbf{y})}$, where h^* was defined previously, and (b) $\epsilon_\Sigma = \sum_{\mathbf{y} \in \mathbf{Y}} \epsilon_b(\mathbf{y})$. Equation 373 follows symmetrically from the expression for error propagation in the product of two real numbers g and h.

□

In deriving BTC or VPIC image-template operations, one encounters the same challenges as described in Section 3.4 for block-encoded imagery. For example, if one has a

von Neumann source template, then one must obtain information from the von Neumann neighbors of the current encoding block, as well as from the block itself. Assuming that n bits are required to encode an edge block, and that the von Neumann topology characterizes the image domain, one has the requirement of 5n bits for the LUT domain size and $n(2^{5n})$ bits for the total LUT size. If n>4, however, the LUT size will be prohibitive for current memory capacities.

Rather than producing large lookup tables that may be impractical to implement, we currently prefer to use application-specific strategies for deriving neighborhood operations over BTC or VPIC encoded imagery. For example, in Chapter 10, we present algorithms for a VPIC edge detector, as well as for morphological operations such as erosion and dilation. These algorithms operate efficiently, and provide insights into the design of algorithms for additional applications. We are currently investigating the possibility of separable lookup tables that would partition the n-bit block descriptor such that several cascaded LUTs could be employed for image-template operations, instead of one large LUT.

9.6. Vector Quantization.

We begin with a brief review of VQ methodology, then progress to pointwise, global-reduce, and image-template operations over VQ-compressed imagery.

9.6.1. Background.

It has been frequently observed that images are comprised of repeated patterns called *texels* [76,77,78]. In an extreme case, a brick wall can be reconstructed from two patterns (brick and mortar texture). Less obvious are the pseudorandom but structured patterns inherent in natural objects such as a carefully mowed lawn or recently tended golf course green, leaves on trees, and ripples on the surface of a pond.

By exploiting spatial redundancy in such patterned imagery, one can generally describe the majority of an image's appearance in terms of a list of exemplar patterns (the codebook) and an array of exemplar indices (the compressed image). The compressed image is merely an array or list of pattern indices, while the decompressed image is obtained by substituting the indexed exemplar patterns according to the map described in the compressed image. In practice, exemplar patterns can be of different size and shape, and may be rotated or scaled anisomorphically. Certain image compression schemes, such as iterative function systems (IFS) based on concepts derived from fractal geometry, extensively exploit recurrent image structure to achieve generally high compression ratios, albeit with representational error [14]. The processing of imagery compressed by IFS-based systems, which are also called *self-VQ*, will be considered in future research.

Returning to more conventional VQ, we note that the exemplar list, also called a *codebook*, is usually constructed under constraint of an error criterion ϵ. As a result, error inherent in the VQ reconstruction step is codebook-dependent. Since the codebook is derived from a given image training set, input nonstationarities may perturb the decompressed image erroneously, thereby resulting in an image representation whose error exceeds ϵ. This lack of determinacy in the reconstruction error can severely impact the accuracy of VQ compressive computation, especially when cascaded operations are employed.

9.6.1.1. Definition. Let $\mathbf{X}$ denote an M×N-pixel array and let $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ denote an image that is tesselated into $k \times l$-pixel encoding blocks $\mathbf{b}(h(\mathbf{x}))$, $\mathbf{x} \in \mathbf{X}$, whose domain is given by $h^*(h(\mathbf{x}))$, where h^* and h were specified in Definition 9.1.1.1. Given a source image $\mathbf{a}$, the VQ transform $T_{VQ}: \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{U}^{\mathbf{Y}} \times (\mathbf{F}^{kl})^{\mathbf{U}}$ clusters the encoding blocks of form $\mathbf{F}^{kl}$ into a codebook $\mathbf{c} \in (\mathbf{F}^{kl})^{\mathbf{U}}$, where $\mathbf{U} \in \mathbf{N}$ is customary. The resultant compressed image $\mathbf{a}_c$ on $\mathbf{Y}$ is $\mathbf{U}$-valued, i.e., references codebook exemplars by their indices.

In this simple formulation of VQ, decompression is merely a matter of substituting exemplars based on indices encoded in $\mathbf{a}_c$. This step requires only $|\mathbf{X}|$ I/O operations, and is thus trivial.

9.6.1.2. Alternative formulation. If the image blocks are normalized to a prespecified mean and standard deviation, then codebook size can be reduced and decompression requires further overhead of $|\mathbf{X}|$ addition operations. For example, let $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ denote an image that is tesselated as described in Definition 9.1.1.2. The following steps are performed for each $\mathbf{y} \in \mathbf{Y}$:

Step 1. Compute the mean μ and standard deviation σ of the block $\mathbf{b}(\mathbf{y})$.

Step 2. Given $\mathbf{x} \in domain(\mathbf{b})$, let $\mathbf{b}$ be thresholded as

$$\hat{\mathbf{b}}(\mathbf{x}) = \begin{cases} \mu - 2\sigma \text{ if } \mathbf{b}(\mathbf{x}) < \mu - 2\sigma \\ \mu + 2\sigma \text{ if } \mathbf{b}(\mathbf{x}) > \mu + 2\sigma \\ \mathbf{b}(\mathbf{x}) \text{ otherwise} \end{cases}, \tag{375}$$

which clamps the extrema of $\mathbf{b}$ to an interval $[\mu-2\sigma,\mu+2\sigma]$.

Step 3. Subtracting the mean, let $\hat{\mathbf{b}} - \mu$ normalized to the interval [-L,L] be correlated with exemplars in a codebook $\mathbf{c} \in (\mathbf{F}^{\mathbf{k}\prime})^{\mathbf{U}}$ to yield a best-match exemplar index $\mathbf{u} \in \mathbf{U}$.

Step 4. Store the mean and exemplar index to yield the following tuple-valued compressed image pixel

$$\mathbf{a}_c(\mathbf{y}) = (\mu, \sigma, \mathbf{u}). \tag{376}$$

Thus, if $\mathbf{F} = \mathbf{R}$, then $\mathbf{a}_c \in (\mathbf{R} \times \mathbf{R}^+ \times \mathbf{U})^{\mathbf{Y}}$.

In practice, the algorithm operates as follows. Under the assumption of normality, Steps 1 and 2 quantize $\mathbf{b}$ into a range that contains 95.4 percent of the values in the real interval $A = [\mu - n\sigma, \mu + n\sigma]$. We set $n = 2$ customarily, since the normal distribution has an associated probability of 0.477 when $n = 2$. Thus, 2(0.477) or approximately 95.4 percent of possible block values are included in A. Step 3 normalizes the block to a prespecified

interval, for efficient comparison with codebook exemplars, and Step 4 constructs the compressed image.

Decompression involves rescaling the $\mathbf{u}$-th exemplar stored in $\mathbf{a}_c(\mathbf{y})$ to the interval $[\mu - 2\sigma, \mu + 2\sigma]$, such that the rescaled block has mean μ. The manipulated block is then substituted into the reconstructed image.

9.6.1.3. Complexity. Step 1 requires work of $|\mathbf{Y}| \cdot (kl - 1)$ additions and $|\mathbf{Y}|$ multiplications to compute the mean, as well as $|\mathbf{Y}| \cdot (2kl - 1)$ additions and $|\mathbf{Y}|$ multiplications and square roots. Step 2 requires $|\mathbf{Y}| \cdot 2kl$ comparison operations, or $|\mathbf{Y}| \cdot kl$ operations if $|range(\mathbf{a})|$ is sufficiently small that table lookup can be employed to compute Equation 375. Steps 3 and 4 require $|\mathbf{Y}|$ additions and I/O operations.

9.6.1.4. Observation. VQ has the following key deficiencies:

1. Codebook generation is difficult computationally, often requiring $\mathbf{O}((|\mathbf{X}|/kl)^2)$ multiplications if the generalized Lloyd clustering algorithm (GLA) is employed [1,12]. The GLA is a descent algorithm with a monotonically decreasing error function that converges by iteratively updating the codebook while attempting to satisfy various centroid and nearest-neighbor conditions. Since the error function is nonconvex and usually contains numerous local minima, the resulting codebook is frequently nonoptimal for a given training set.
2. The VQ codebook must be regenerated for each image test set and is sensitive to input nonstationarities. In contrast, VPIC's codebook is data-invariant [9], being dependent only on knowledge about the human visual system (HVS).
3. VQ's codebook is customarily computed under a mean-square or Hamming-distance error criterion that may result in obscuration of image features that are useful for ATR. In particular, VQ has no ability to discriminate between spatial

attributes of various image features, which can lead to significant errors when examining images for the presence of small features (i.e., whose spatial frequency is less than k/M or l/N.

4. Codebook search overhead is large, typically requiring a minimum of $|\mathbf{U}| \cdot \log(|\mathbf{U}|)$ comparison operations.

Alternatives to the GLA abound in the literature. For example, Kohonen implemented a neural-network based algorithm that approximates least-mean-squares (LMS) clustering, with dynamic update [79]. Kohonen's LMS net circumvents the problem of convergence to a local minimum via network design that purportedly ensures convergence to a global minimum. A further advantage of Kohonen's method is the admission of input nonstationarities via dynamic codebook updating as each source vector is presented. Unfortunately, training times for the neural net are typically long. Additional variants of the Lloyd algorithm have been proposed that utilize stochastic clustering techniques to produce a slightly more compact codebook than that which results from Lloyd's greedy algorithm [80]. Codebook size can also be varied by increasing the error constraint, albeit at the cost of visual fidelity in the decompressed image.

As a result of the foregoing deficiencies, we present the following discussions of information loss and computational error inherent in VQ compression.

9.6.1.5. Information loss. By design, vector quantization produces a codebook whose exemplars represent source image blocks under an MSE criterion ϵ. Using notation employed in the VPIC transform description of Section 9.5.2, each source block $\mathbf{b}(\mathbf{y})$, $\mathbf{y} \in \mathbf{Y}$, maps to its respective exemplar $\mathbf{c}(\mathbf{u})$, where $\mathbf{u} \in \mathbf{U}$, under the constraint

$$\| \mathbf{b}(\mathbf{y}) - \mathbf{c}(\mathbf{u}) \| < \epsilon , \tag{377}$$

which ensures that the blockwise error remains within the prespecified bound $\pm\epsilon$. This means that the total absolute error ϵ_t in representing the source image will be bounded

as $\epsilon_t \leq |\mathbf{Y}| \cdot \epsilon$. As a result, the mean error per pixel in the reconstructed source image on $\mathbf{X}$ is given by

$$\overline{\epsilon_{VQ}} \leq \frac{|\mathbf{Y}|}{|\mathbf{X}|} \cdot \epsilon = \frac{\epsilon}{CR_D}. \tag{378}$$

9.6.1.6. Remark. Although Equation 378 initially appears to state that the per-pixel information loss ϵ_{VQ} decreases as the compression ratio increases, this is not the case, since the block representation error ϵ increases with CR_D as fast as (or faster than) ϵ_{VQ} would decrease. If this were not so, then there would (in principle) exist some compression ratio at which ϵ_{VQ} would asymptotically approach zero. Unfortunately, the converse occurs in practice, i.e., ϵ_{VQ} grows with CR due to codebook compression. In the presence of static encoding block size, CR_D remains unchanged.

9.6.2. Pointwise Arithmetic.

Pointwise VQ operations can be quite simple, but raise questions concerning tradeoffs between the propagation of representational error and codebook growth.

9.6.2.1. Observation. We have found that pointwise arithmetic operations $\bigcirc : \mathbf{F}^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ have analogues over imagery compressed by a VQ transform $T_{VQ} : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{U}^{\mathbf{Y}} \times (\mathbf{F}^{\mathbf{kl}})^{\mathbf{U}}$ can be derived merely by applying $\bigcirc$ to codebook exemplars indexed in $\mathbf{U}^{\mathbf{Y}}$. However, this observation requires that one of the following three methods is employed in deriving $\bigcirc' : \mathbf{G}^{\mathbf{Y}} \times \mathbf{G}^{\mathbf{Y}} \rightarrow \mathbf{G}^{\mathbf{Y}}$, the analogue of $\bigcirc$ over $range(T_{VQ})$:

Method 1. After Cosman et al. [1,12], if $\mathbf{b}$ and $\mathbf{d}$ are two encoding blocks in $domain(\bigcirc)$ and $\mathbf{c_b}$, $\mathbf{c_d}$ are the corresponding codebooks in $p_2(range(T_{VQ}))$, then we first locate the indices $\mathbf{u_b}$ and $\mathbf{u_d}$ in $\mathbf{U}$ of the exemplars that match $\mathbf{b}$ and $\mathbf{d}$ within a prespecified error threshold ϵ. Such indices are found in the compressed images $\mathbf{b_c} = T_{VQ}(\mathbf{b})$ and $\mathbf{d_c} = T_{VQ}(\mathbf{d})$. By forming a lookup table L whose

domain consists of the cross product of exemplars in $\mathbf{c_b}$ and $\mathbf{c_d}$, we obtain a new codebook $\mathbf{c_e}$, which has an entry that corresponds to the block $\mathbf{e} = \bigcirc(\mathbf{b},\mathbf{d})$.

Method 2. We have modified Method 1 to reduce codebook growth, as follows. Letting operands be defined as in Method 1, we have some codebook $\mathbf{c_e}$ that is the union of $\mathbf{c_b}$ and $\mathbf{c_d}$. We compute $\mathbf{e} = \bigcirc(\mathbf{b},\mathbf{d})$, then find the exemplar in $\mathbf{c_e}$ that best matches $\mathbf{e}$, although the resultant error may exceed ϵ. The index $\mathbf{u_e}$ of this exemplar is the corresponding value in the compressed image $\mathbf{e_c} = \bigcirc(\mathbf{b_c},\mathbf{d_c})$.

Method 3. A further modification of Method 1 yields constant codebook size. Letting operands be defined as in Method 1, we assume that only one codebook $\mathbf{c}$ exists whose exemplars encode blocks $\mathbf{b}$ and $\mathbf{d}$ as indices $\mathbf{u_b}$ and $\mathbf{u_d}$ in $\mathbf{U}$. Then, the computed block $\mathbf{e} = \bigcirc(\mathbf{b},\mathbf{d})$ is encoded in terms of the best-match exemplar in $\mathbf{c}$ whose index $\mathbf{u_e}$ is substituted in $\mathbf{e_c}$.

9.6.2.2. Remark. The preceding methods have the following implementational advantages and disadvantages.

Method 1. Advantage. The error with which the result $\mathbf{e}$ is represented is minimized, since a large codebook $\mathbf{c_e}$ is constructed from the exemplars in $\mathbf{c_b}$ and $\mathbf{c_d}$. Thus, all combinations of the source exemplars are accounted for.

Disadvantage. Codebook size is large, and codebook growth is exponential in the number of cascaded operations. For example, if there are M exemplars in $\mathbf{c_b}$ and $\mathbf{c_d}$, then $\mathbf{c_e}$ can have M^2 exemplars. Given L cascaded operations, M^{2L} exemplars are possible in the L-th codebook. This is clearly unacceptable for practical compression applications.

Method 2. Advantage. Codebook size is reduced from $|\mathbf{c_b}| \cdot |\mathbf{c_d}|$ to $|\mathbf{c_b} \cup \mathbf{c_d}|$, which is a significant savings. That is, if $\mathbf{c_b}$ and $\mathbf{c_d}$ are derived from similar training sets, then they are likely to have similar exemplars. In practice, codebook growth is usually sublinear in L for images of natural scenes containing a small

number of manufactured objects.

Disadvantages. The reduction in codebook size implies a compromise that yields increased error, since the operation $\mathbf{e}_c = \bigcirc(\mathbf{b}_c,\mathbf{d}_c)$ is no longer represented uniquely in $\mathbf{c}_e$. Additionally, the codebook still continues to grow with the number of cascaded VQ-based operations.

Method 3. Advantage. Codebook size remains constant as a function of the number of operands of $\bigcirc$ and the number of operations L.

Disadvantage. Error accumulation can be unacceptably high, since the single codebook $\mathbf{c}$ is likely not equal to $\mathbf{c}_b \cup \mathbf{c}_d$, and thus does not contain all exemplars in $\mathbf{c}_b$ and $\mathbf{c}_d$. As a result, the error with which $\mathbf{b}_c$ and $\mathbf{d}_c$ represent $\mathbf{b}$ and $\mathbf{d}$ is higher than in Methods 1 or 2. When such error is propagated through $\bigcirc$, then the result of $\bigcirc$ is quantized by representation with an exemplar in $\mathbf{c}$, higher error results. This is due to the fact that the error ϵ_{VQ} that is input to $\epsilon_{\bigcirc}$ is larger than in Methods 1 or 2.

We discuss issues of computational error and information loss, as follows.

9.6.2.3. Information loss. Let Observation 9.6.2.1 hold. In Methods 1–3, there are three sources of error (in principle), namely, (1) the encoding error ϵ_{VQ} with which $\mathbf{b}$ and $\mathbf{d}$ are represented by $\mathbf{b}_c$ and $\mathbf{d}_c$; (2) the computational error inherent in the pointwise operation $\bigcirc$, which is denoted by $\epsilon_{\bigcirc}$; and (3) the error with which the result $\mathbf{e} = \bigcirc(\mathbf{b},\mathbf{d})$ is encoded in the compressed result $\mathbf{e}_c$. The resultant error specific to an analogous operation $\bigcirc'$, which we denote by $\epsilon_{\bigcirc'}$, has the following general formulation:

$$\epsilon_{\bigcirc'} \leq \epsilon_{VQ}(\epsilon_{\bigcirc}[\epsilon_{VQ}(\epsilon_b),\epsilon_{VQ}(\epsilon_d)]) . \tag{379}$$

If $\bigcirc = +$, then $\bigcirc' = +$ and $\epsilon_{\bigcirc'}$ is given by

$$\epsilon_{\bigcirc'} \leq \epsilon_{VQ}(\epsilon_{VQ}(\epsilon_b) + \epsilon_{VQ}(\epsilon_d)) \approx 3 \cdot \epsilon_{VQ} , \tag{380}$$

where the approximation is formulated under the realistic assumption that ϵ_{VQ} is constant for codebooks $\mathbf{c_b}$, $\mathbf{c_d}$, and $\mathbf{c_e}$ if Method 1 is employed.

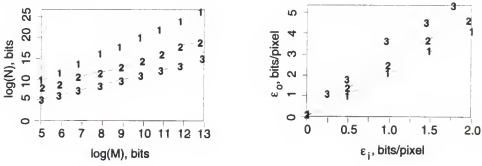
Since Method 1 forms the codebook $\mathbf{c_e}$ from the operand codebooks $\mathbf{c_b}$ and $\mathbf{c_d}$, representational error is less than in Methods 2 and 3. However, this assumes that the statistics of $\mathbf{e}$ upon which $\mathbf{c_e}$ is based are preserved when $\bigcirc$ is applied to $\mathbf{c_b}$ and $\mathbf{c_d}$. With linear operations $\bigcirc$, input statistics (of $\mathbf{b}$ and $\mathbf{d}$) are preserved in $\mathbf{e}$, due to the linearity property of expected values. However, such is not the case for nonlinear operations such as $\log_{\mathbf{b}}(\mathbf{d})$.

Since Method 2 quantizes the result $\mathbf{e}$ into an exemplar in $\mathbf{c_b} \cup \mathbf{c_d}$, if the statistics of $\mathbf{b}$ and $\mathbf{d}$ are preserved in $\mathbf{e}$, then there is no additional representation error beyond that incurred in Method 1. In practice, we have found that, for most pointwise operations (particularly transcendental functions), $\mathbf{c_b} \cup \mathbf{c_d}$ may poorly represent blocks that have values near the extrema of $range(\mathbf{e})$. The reasons for this behavior, which are being considered in ongoing research, likely derive from the LMS clustering algorithm used in the generalized Lloyd clustering algorithm with which $\mathbf{c_b}$ and $\mathbf{c_d}$ are formed. That is, the LMS algorithm appears to produce clusters less accurately when cluster size is small, as occurs near range extrema that occur infrequently in training set imagery.

In Method 3, a single codebook $\mathbf{c}$ is employed without regard to differences in statistics among $\mathbf{b}$, $\mathbf{d}$, and $\mathbf{e}$. As a result, all ϵ_{VQ} could be higher than in Methods 1 or 2, thereby yielding high $\epsilon_{\bigcirc'}$. As a result, we prefer to employ Method 3 only when constant codebook size is required. Method 2 presents a useful compromise between codebook growth and error accumulation.

9.6.2.4. Example. We computed codebook growth and error for VQ codebooks derived from over 100 natural images of outdoor scenes containing small, man-made targets. Figure 16a (b) illustrates the differences in codebook size (error) for the original images and the images subjected to addition using Methods 1–3. Note that the codebook grows maximally

for Method 1, and grows least for Method 2, as predicted by theory. Constant codebook size is exhibited by Method 3. Note that the error growth of Method 1 is typical for addition, due to the additive error propagation law. Method 3 did not lend itself to zero codebook error, hence that data point is not represented in Figure 16b. Note that the results shown in the following figure vary with (a) the type of operation employed, (b) the image data, and (b) the type of codebook clustering algorithm employed (in this case, a textbook LMS clustering algorithm).



(a) (b)

Figure 16. Codebook growth and error as a function of codebook size and Methods 1–3 for VQ pointwise addition: (a) input and output code book sizes M and N, (b) input and output errors ϵ_i and ϵ_o.

9.6.3. Global Reduce Operations.

9.6.3.1. Observation. VQ imagery has attractive properties for computing global-reduce operations. For example, if one has the sum of each exemplar stored in an array of partial sums, then the VQ image histogram , which describes the probability of a given exemplar, also describes the frequency-of-occurrence of each partial sum. It is then a simple matter to additively combine the partial sums, weighted by their frequency-of-occurrence. Similarly, the image maximum is the maximum of all exemplar maxima, and symmetrically for the minimum.

The following theorem is illustrative.

9.6.3.2. Theorem. If Assumption 9.5.4.2 holds and T is a vector quantization transform that produces a compressed image $\mathbf{a_c} \in \mathbf{U^Y}$ and a codebook $\mathbf{c} \in \left(\mathbf{F}^{kl}\right)^{\mathbf{U}}$ from a source image $\mathbf{a} \in \mathbf{F^X}$ (per Definition 9.6.1.1), then duals of the global reduce operations $\mathbf{\Gamma} \in \{\Sigma, \vee, \Pi\}$ are given by:

$$\mathbf{\Gamma}'(\mathbf{a_c}) = \underset{\mathbf{y} \in domain(\mathbf{a_c})}{\mathbf{\Gamma}} \mathbf{\Gamma}(\mathbf{c}(\mathbf{a_c}(\mathbf{y}))). \tag{381}$$

Additionally, if $\mathbf{h} \in \mathbf{N^U}$ denotes the histogram of a compressed image $\mathbf{a_c} \in \mathbf{U^Y}$, then the following dual formulations hold:

$$\begin{aligned} \Sigma\, \mathbf{a} &\approx \sum_{\mathbf{u} \in domain(\mathbf{c})} \mathbf{h}(\mathbf{u}) \cdot \mathbf{\Sigma}\, \mathbf{c}(\mathbf{u}) \\ \vee \mathbf{a} &\approx \bigvee_{\mathbf{u} \in domain(\mathbf{c})} \vee \mathbf{c}(\mathbf{u}) \\ \Pi\, \mathbf{a} &\approx \prod_{\mathbf{u} \in domain(\mathbf{c})} (\mathbf{\Pi}\, \mathbf{c}(\mathbf{u}))^{\mathbf{h}(\mathbf{u})}. \end{aligned} \tag{382}$$

Proof. The proof is outlined as follows. Assuming that the givens hold, the proof follows directly from the fact that the binary operation γ, the basis function of $\mathbf{\Gamma}$, is associative and commutative. Additionally, the approximation results from the information loss inherent in VQ.

□

Image-template operations over VQ imagery have associated implementational difficulties that are similar to those encountered in processing BTC and VPIC-encoded imagery. Per the comments at the end of the preceding section, we are currently investigating VQ edge detection and morphological erosion operations, and discuss such methods at a high level in the following chapter.

CHAPTER 10
APPLICATIONS OF COMPRESSIVE IMAGE PROCESSING

Having discussed the BTC, VPIC, VQ, and JPEG transformations, we next describe several compressive operations that are useful in target recognition practice. We begin with smoothing via local averaging.

10.1. Compressive Smoothing

Local averaging is customarily employed in reducing the visual effect of noise, to increase the definition of targets against natural backgrounds. Such averaging also helps reduce effects such as variance that results from pseudorandom textures (e.g., grass and leaves) in images of natural scenes.

10.1.1. Compressive Local Averaging.

For purposes of discussion, we consider local averaging of imagery using a unitarily-valued Moore or von Neumann template.

10.1.1.1. Concept. Image smoothing can be approximated by compressive smoothing within an encoding block via replicating the block mean at each pixel of a given block. Then, we convolve a blur function (e.g., a unitarily-valued von Neumann template) with the compressed image to obtain smoothed block boundaries. A key postprocessing step involves horizontal (vertical) one-dimensional unitary von Neumann template to smooth vertical (horizontal) block boundaries, followed by a two-dimensional unitary von Neumann template smooth pixels, where the source point of such a pixel is the intersection of adjacent encoding blocks.

10.1.1.2. Algorithm. Let a source image $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ be encoded by a transform $T: \mathbf{F}^{\mathbf{X}} \to \mathbf{G}^{\mathbf{Y}}$ that tessellates $\mathbf{a}$ into $k \times l$-pixel blocks $\mathbf{b}(\mathbf{y}) \in \mathbf{F}^{kl}, \mathbf{y} \in \mathbf{Y}$. Further assume that T preserves in the compressed image $\mathbf{a}_c = T(\mathbf{a})$ the block mean μ as $p_1(\mathbf{a}_c)$. Let the spatially invariant templates $\mathbf{r},\mathbf{s},\mathbf{t} \in (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}}$ be defined by their weights as follows:

$$\mathbf{r} = (1 \quad 1 \quad 1), \quad \mathbf{s} = \mathbf{r}', \quad \text{and} \quad \mathbf{t} = \begin{pmatrix} & 1 & \\ 1 & 1 & 1 \\ & 1 & \end{pmatrix}, \tag{383}$$

where prime ($'$) denotes transpose and the target point is italicized. Let $\mathbf{r}_c,\mathbf{s}_c,\mathbf{t}_c \in (\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}$ be compressive templates whose weight matrices are identical to those of $\mathbf{r}$, $\mathbf{s}$, and $\mathbf{t}$, respectively. Note that the preceding templates need not be limited in size to an $m \times n$–pixel domain, where $m,n = 3$. Rather, m or n can be as large as $\wedge(\lfloor k/2 \rfloor, \lfloor l/2 \rfloor)$, since the block mean can be assigned only to the central region of the source block domain.

We simulate the action of $\bigcirc: \mathbf{F}^{\mathbf{X}} \times (\mathbf{F}^{\mathbf{X}})^{\mathbf{X}} \to \mathbf{F}^{\mathbf{X}}$ over *range*(T) using a dual compressive operation $\Omega: \mathbf{G}^{\mathbf{Y}} \times \left[(\mathbf{F}^{\mathbf{Y}})^{\mathbf{Y}}\right]^3 \to \mathbf{F}^{\mathbf{X}}$ such that

$$\mathbf{b} = \Omega(\mathbf{a}_c, \mathbf{r}_c, \mathbf{s}_c, \mathbf{t}_c) \approx \bigcirc(\mathbf{a}, \mathbf{t}), \tag{384}$$

which is computed via the following algorithm.

Let the indexing functions h and h^* be as given in Definition 9.1.1.1, and let the indexing functions $c,r,j: \mathbf{Y} \to 2^{\mathbf{X}}$ respectively return the source neighborhood associated with the column, row, and row-column intersection boundaries of the $\mathbf{y}$-th encoding block.

Step 1. Let the column averages be stored in $\mathbf{d}_c = p_1(\mathbf{a}_c) \oplus \mathbf{r}_c$.

Step 2. Similarly, the row averages are stored in $\mathbf{d}_r = p_1(\mathbf{a}_c) \oplus \mathbf{s}_c$.

Step 3. Store the row-column intersection averages in $\mathbf{d}_j = p_1(\mathbf{a}_c) \oplus \mathbf{t}_c$.

Step 4. For each $\mathbf{y} \in \mathbf{Y}$ do:

$$\begin{aligned} \mathbf{b}|_{h^*(\mathbf{y})} &= p_1(\mathbf{a}_c(\mathbf{y})) \\ \mathbf{b}|_{c(\mathbf{y})} &= \mathbf{d}_c(y) \\ \mathbf{b}|_{r(\mathbf{y})} &= \mathbf{d}_r(y) \\ \mathbf{b}|_{j(\mathbf{y})} &= \mathbf{d}_j(y). \end{aligned} \tag{385}$$

The algorithm operates as follows. First, the local averages are taken at block boundaries. This can be done via the compressive convolution shown in Steps 1–3. In Step 4, we merely input the block means to the non-boundary areas of each block domain $h^*(\mathbf{y})$, then input boundary values at points given by c, r, and j.

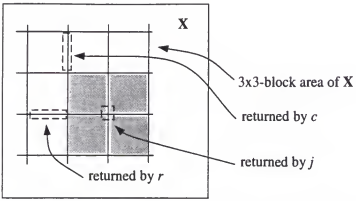


Figure 17. Block configuration for compressive local averaging, showing block neighborhoods (dashed boxes) on $\mathbf{X}$ returned by c, r, and j.

10.1.1.3. Observation. As shown in Part 1, if we have an image smoothing operation $\bigcirc$ and its compressive dual Ω, we can define a dual CCS

$$X = \left(\left\{\mathsf{F}^{\mathbf{X}}, (\mathsf{F}^{\mathbf{X}})^{\mathbf{X}}; \bigcirc\right\}, \{T, U; \Theta\}, \left\{\mathsf{G}^{\mathbf{Y}}, (\mathsf{F}^{\mathbf{Y}})^{\mathbf{Y}}; \Omega\right\}\right), \tag{386}$$

where T was given in the preceding section and U is an identity transform on the weight matrices of templates $\mathbf{r},\mathbf{s},\mathbf{t} \in (\mathsf{F}^{\mathbf{X}})^{\mathbf{X}}$ that produces templates $\mathbf{r}_c,\mathbf{s}_c,\mathbf{t}_c \in (\mathsf{F}^{\mathbf{Y}})^{\mathbf{Y}}$.

10.1.1.4. Complexity and information loss. Steps 1–3 require $\mathbf{O}(|\mathbf{Y}|)$ multiplications and additions or, more precisely, $11 \cdot |\mathbf{Y}|$ multiplications and $8 \cdot |\mathbf{Y}|$ additions. The definition of h^* can be modified slightly, such that Step 4 requires $|\mathbf{X}|$ I/O operations when block means are placed in non-boundary regions only. Neglecting I/O cost, the total complexity of compressive smoothing is given by

$$\begin{aligned} W_{\text{smooth}}(\mathbf{Y}) = {} & |\mathbf{Y}| \cdot [(|\mathcal{S}(\mathbf{r}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{s}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{t}_c(\mathbf{y}))|) \text{ multiplications} \\ & + [(|\mathcal{S}(\mathbf{r}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{s}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{t}_c(\mathbf{y}))| - 3) \text{ additions}], \mathbf{x} \in \mathbf{X}, \mathbf{y} \in \mathbf{Y}. \end{aligned} \tag{387}$$

Neglecting possible differences between the number of multiplications and additions per block, the corresponding sequential computational speedup inherent in X can be approximated as

$$\eta(X) \approx \frac{|\mathbf{X}| \cdot (|\mathcal{S}(\mathbf{t_x})|) \cdot (\triangle t_m + \triangle t_a)}{|\mathbf{Y}| \cdot (|\mathcal{S}(\mathbf{r_c(y)})| + |\mathcal{S}(\mathbf{s_c(y)})| + |\mathcal{S}(\mathbf{t_c(y)})|) \cdot (\triangle t_m + \triangle t_a)} = CR_d \cdot \frac{(|\mathcal{S}(\mathbf{t_x})|)}{(|\mathcal{S}(\mathbf{r_c(y)})| + |\mathcal{S}(\mathbf{s_c(y)})| + |\mathcal{S}(\mathbf{t_c(y)})|)}, \tag{388}$$

where $\triangle t_m$ and $\triangle t_a$ denote the delay incurred by a multiplication or addition operation. Thus, X exhibits a speedup of $\mathbf{O}(CR_d)$ or greater if the ratio of the total size of the compressive templates to the size of the uncompressed template is less than unity. For example, in Section 10.1.1.2, $\eta = CR_d \cdot 8/11$. If we take the convenient case of a 4×4-pixel encoding block, where $CR_d = 16$, then $\eta = 11.6$, a nontrivial speedup.

10.1.1.5. Information loss. Note that $\bigcirc'$ represents the convolution of **a** with **t** by assigning block means μ to the central region of each block and local averages to the boundary regions. Thus, $\bigcirc'$ erroneously represents the local average in the center of the block if μ does not represent the local average. Similarly, the boundary averages are taken across the block means, and thus erroneously represent boundary values. The representational error, which is dependent on the relationship between template size and weight as well as blocksize, is also dependent upon the template configuration (i.e., von Neumann or Moore templates).

For example, if **t** is 5 pixels square and $k,l = 5$, then there will be a 2–pixel wide region on either side of the encoding block boundaries that is portrayed incorrectly if the size of $\mathbf{t_c}$ is less than 5×5 pixels. If **t** is a unitarily-valued Moore template, then the mean of an encoding block that is the same size as **t** clearly portrays the block mean. In the remaining cases, which appear to predominate in practice, the block mean does not portray the local average at the center of the encoding block. Thus, we specify U as the identity transform applied to the weight matrices of **r**, **s**, and **t**, in order to preserve the size of **t** in $\mathbf{t_c}$.

10.1.1.6. Observation. An additional problem with compressive smoothing is that the weights of $\mathbf{r}$, $\mathbf{s}$, and t must be preserved in $\mathbf{r}_c$, $\mathbf{s}_c$, and t_c in order that the boundary averages can be computed over the compressed image in the same manner as averaging would be performed in the image smoothing operation. The formulation of U as an identity transformation ensures this result.

We next provide an example of compressive smoothing applied to multi-target imagery taken against a grassy background.

10.1.1.7. Example. Let the source image $\mathbf{a}$ shown in Figure 18a (i.e., a field of small targets) be convolved with the unitary von Neumann template $\mathbf{t}$ to yield the image of Figure 18b. Let $\mathbf{a}$ be compressed by BTC (4×4-pixel blocks) to yield the mean image shown in Figure 18c. After the compressive smoothing algorithm is applied to 18c, the image of Figure 18d results.

As can be seen from the figure, the visual effect of compressive smoothing on high-resolution imagery yields slightly increased blur with respect to noncompressive smoothing when the blocksize is larger than the template dimension. This effect can be corrected by using a larger template, which decreases computational efficiency. In this example, a nontrivial speedup of $\eta = 10.6$ was achieved on a Sun Sparcstation-10 running Sun FORTRAN-77 on SunOS Version 4.1. This efficiency is clearly less than CR_D, due to the overhead (in addition to the $\mathbf{O}(CR_D$ cost of local averaging) incurred by the boundary-smoothing templates. When considering the more involved operation of VPIC-based edge detection, we shall see that computational efficiencies that approach CR_d are achievable in practice.

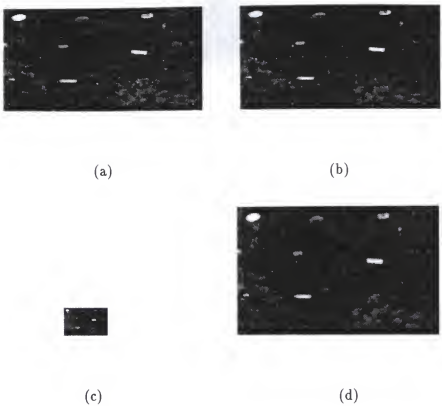


Figure 18. Example of compressive smoothing at $CR_d = 16{:}1$: (a) source image, (b) locally averaged image using unitary von Neumann template, (c) compressed array of block means over which processing actually occurs, (d) decompressed result of compressive smoothing.

We next consider the problem of smoothing VQ-compressed imagery.

10.1.2. VQ-based Compressive Smoothing.

We begin by noting several useful properties of the VQ codebook exemplars.

10.1.2.1. Observation. Recall that the VQ codebook **c** contains exemplars that represent one or more encoding blocks of the source image. This implies two strategies for simulating image smoothing over VQ-compressed imagery. First, one can alter the codebook by taking the block mean, then reconstructing the VQ-compressed image and applying boundary

smoothing operations similar to those described in Section 10.1.1.2. The boundary smoothing ameliorates the "blocking effect" that is frequently found to be objectionable in VQ reconstructions.

A second method is to merely smooth the codebook exemplars and substitute them into the reconstructed image. This has the advantage of preserving smoothed features within-block, but also has the disadvantage of exaggerating the blocking effect due to the presence of boundary effects resulting from the local (blockwise) smoothing operation. Given a codebook of size M, both methods have $\mathbf{O}(M)$ overhead in codebook processing. However, the first method has an additional $\mathbf{O}(|\mathbf{Y}|)$ overhead, where $\mathbf{Y}$ denotes the compressed domain. Given the preceding disadvantages, we developed a third method of VQ-based smoothing, described in the following section, which combines the best features of the preceding methods.

10.1.2.2. Algorithm. Given a source image $\mathbf{a}$ on $\mathbf{X}$ that is transformed by a VQ transform T to yield a compressed image $\mathbf{a}_c$ on $\mathbf{Y}$ and codebook $\mathbf{c}$, we perform the following steps to compute an improved dual of compressive smoothing:

Step 1. Apply a smoothing operation directly to each exemplar in $\mathbf{c}$, thereby producing a new codebook $\mathbf{d}$ that contains smoothed exemplar blocks. Thus, $\mathbf{a}_c$ portrays a smoothed version of $\mathbf{a}$, but has blocking effects due to boundary effects resulting from the smoothing.

Step 2. Implement the source image reconstruction (decompression) step to produce image $\mathbf{b}$ on *domain*($\mathbf{a}$) by substituting exemplars from $\mathbf{d}$ indexed in $\mathbf{a}_c$.

Step 3. Apply boundary smoothing templates $\mathbf{r}_c$, $\mathbf{s}_c$, and $\mathbf{t}_c$ defined in Section 10.1.1.2 to the encoding block boundaries of $\mathbf{b}$ to reduce boundary and blocking effects.

In contrast to the preceding method of computing block means, within-block features are preserved and between-block variance is smoothed.

10.1.2.3. Complexity and information loss. Given a codebook of size M and a smoothing template $\mathbf{t}$, Step 1 of the preceding algorithm requires $M \cdot |\mathcal{S}(\mathbf{t})|$ multiplications and $M \cdot (|\mathcal{S}(\mathbf{t})| - 1)$ additions. Letting $\mathbf{X} = domain(\mathbf{a})$, Step 2 requires $|\mathbf{X}|$ I/O operations. Step 3 requires one application each of $\mathbf{r}_c$ and $\mathbf{s}_c$ as well as two applications of $\mathbf{t}_c$ per pixel in $\mathbf{Y}$. The total work of compressive smoothing using VQ imagery is therefore given by:

$$\begin{aligned} W_{VQsm}(\mathbf{Y}, M, k, l) &= Mkl \cdot |\mathcal{S}(\mathbf{t_x})| \\ &\quad + |\mathbf{Y}| \cdot (|\mathcal{S}(\mathbf{r}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{s}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{t}_c(\mathbf{y}))|) \text{ multiplications} \\ &\quad + Mkl \cdot (|\mathcal{S}(\mathbf{t_x})| - 1) + |\mathbf{Y}| \cdot (|\mathcal{S}(\mathbf{r}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{s}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{t}_c(\mathbf{y}))| - 3) \text{ additions}, \end{aligned} \tag{389}$$

where $\mathbf{x} \in \mathbf{X}$ and $\mathbf{y} \in \mathbf{Y}$. Neglecting the difference between the number of multiplications and additions per block, the corresponding sequential computational speedup is given by:

$$\begin{aligned} \eta &\approx \frac{|\mathbf{X}| \cdot (|\mathcal{S}(\mathbf{t_x})|) \cdot (\Delta t_m + \Delta t_a)}{Mkl \cdot (|\mathcal{S}(\mathbf{t_x})|) + |\mathbf{Y}| \cdot (|\mathcal{S}(\mathbf{r}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{s}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{t}_c(\mathbf{y}))|) \cdot (\Delta t_m + \Delta t_a)} \\ &= CR_D \cdot \frac{(|\mathcal{S}(\mathbf{t_x})|)}{\frac{Mkl}{|\mathbf{Y}|} \cdot (|\mathcal{S}(\mathbf{t_x})|) + (|\mathcal{S}(\mathbf{r}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{s}_c(\mathbf{y}))| + |\mathcal{S}(\mathbf{t}_c(\mathbf{y}))|)}, \mathbf{x} \in \mathbf{X}, \mathbf{y} \in \mathbf{Y}. \end{aligned} \tag{390}$$

Comparing VQ- and VPIC-based smoothing, we note that the crucial difference between the preceding equation and Equation 388 is the additive constant $c = \frac{Mkl}{|\mathbf{Y}|} \cdot (|\mathcal{S}(\mathbf{t_x})|)$ in the denominator of η. If $|\mathbf{Y}| \gg M$, which is usually the case, then c is small and does not substantially affect η.

10.1.2.4. Example. If $\mathbf{X}$ is a 1024×1024-pixel array and $k,l = 4$, then $|\mathbf{Y}| = 64K$. Customary values for M range between 64 and 256 exemplars. Thus, $1024 \leq Mkl \leq 4096$ and $0.0625 \leq c \leq 0.156$. In the example of BTC smoothing efficiency, $\eta = CR_D \cdot 8/11$, which implies that $\eta = 11.6$ would be perturbed by c to yield $11.44 \leq \eta \leq 11.53$, a difference of approximately one percent of η, which is not significant.

The discussion of information loss presented in Section 10.1.1.4 also pertains to VQ-based smoothing.

10.2. Compressive Edge Detection.

Various properties of compressed image formats lend themselves to exploitation in image segmentation tasks. In this case, we examine techniques by which the edge-preserving properties of BTC, VPIC, JPEG, and VQ transforms can be employed to simulate edge detection over compressed imagery. This approach is useful for variance-based target classification [62,81].

10.2.1. BTC-based Edge Detection.

10.2.1.1. Observation. The BTC transform discussed in Chapter 9 computes the mean and standard deviation σ (gradient g) of an encoding block as a measure of block variance. The rationale for this computation is based on the fact that blocks which contain more edge information (i.e., stronger edge transitions) exhibit greater variance. By thresholding the compressed image based on σ or g, then combining stored bitmap patterns with σ or g, BTC- and VPIC-compressed imagery can be efficiently manipulated to approximate edge imagery in the source domain. We thus specify the following dual operations in blockwise fashion.

10.2.1.2. Algorithm. Let an image $\mathbf{a}$ on $\mathbf{X}$ be compressed by T = BTC to yield a compressed image $\mathbf{a}_c$ on $\mathbf{Y}$. The compressed pixel values have form $(\mu,\sigma,\mathbf{d})$, that denote the mean, standard deviation, and zero-crossing bitmap of the corresponding encoding block in $\mathbf{a}$. If a mean block is portrayed by a compressed pixel value $\mathbf{a}_c(\mathbf{y})$, where $\mathbf{y}$ is a point in $\mathbf{Y}$, then σ and $\mathbf{d}$ are zero-valued.

Compressive edge detection using BTC imagery requires that the bitmap $\mathbf{d}$ be transformed using an edge detection operation e: $\{0,1\}^{kl} \rightarrow \{0,1\}^{kl}$ that computes edge detection over bits in the interior of the block and conditionally alters bits on block boundaries.

For example, the mapping

$$e: \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \mapsto \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \tag{391}$$

inverts the bit in row 3, column 2 that does not touch the block boundary. The reason for this discrimination between interior and boundary information pertains to the fact that the upper boundary of a given block (called $\mathbf{b}_1$, for convenience) may be adjacent to a block $\mathbf{b}_2$ that has unitary bits which are adjacent to unitary bits in $\mathbf{b}_1$. In such cases, an edge may not be present at a block boundary that is subtended by a unitarily-valued region. For example, consider the following juxtaposition of blocks and the corresponding edge detection:

$$e: \left(\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \mapsto \left(\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right). \tag{392}$$

Note that the block boundary (denoted by a dashed line) is intersected by a horizontal region or *run* of three unitary bits. Due to the edge detection algorithm for this type of block, which is based (in this case) upon Moore connectivity, the boundary crossing region, which is formerly configured as $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, becomes $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. In contrast, given von Neumann connectivity, the run of pixels that spans the block boundary would be preserved. We also note that, following edge detection, e performs the bipolar transform on all bits of the edge-detected bitmap $\mathbf{d}$, such that $b(\mathbf{d}) = 2\mathbf{d}\text{-}1$. The design of e for VPIC-compressed imagery is discussed later in this chapter.

Given the preceding operations, compressive edge detection over BTC imagery is achieved by performing the following steps:

Step 1. Initialize a decompressed image on $\mathbf{X}$ denoted by $\mathbf{b} = \mathbf{0}$.

Step 2. For each $\mathbf{y} \in \mathbf{Y}$, do:

$$\begin{aligned} &\texttt{if } (p_2(\mathbf{a}_c(\mathbf{y})) = 0) \texttt{ then } \mathbf{b}|_{h^*(\mathbf{y})} = 0 \\ &\texttt{else} \\ &\quad \mathbf{b}|_{h^*(\mathbf{y})} = p_2(\mathbf{a}_c(\mathbf{y})) \cdot e[p_3(\mathbf{a}_c(\mathbf{y}))] \\ &\texttt{endif}, \end{aligned} \tag{393}$$

where h^* was defined previously.

Thus, mean blocks are zeroed while edge blocks are treated as edge maps that are multiplied by the edge magnitude, which we previously showed is approximated by the standard deviation σ.

10.2.1.3. Complexity. Given a total of $|\mathbf{Y}|$ blocks, of which a fraction r are edge blocks, then the preceding BTC-based edge detection algorithm requires $(r-1) \cdot |\mathbf{Y}| \cdot k$*l* I/O operations to set mean blocks to zero, $r \cdot |\mathbf{Y}|$ invocations of e, and $r \cdot |\mathbf{Y}| \cdot k$*l* sign changes to set the edge magnitudes to $\pm\sigma$. Assuming that the I/O overhead and sign change (a one-bit operation) can be neglected, the overhead of edge detection is primarily due to $r \cdot |\mathbf{Y}|$ invocations of e. Our current implementation of e requires at least $2k$*l* comparison operations for each block. Thus, the cost of BTC-based edge detection is given by:

$$W_{\text{BTCed}}(\mathbf{Y}, r) \geq 2r \cdot |\mathbf{Y}| \cdot CR_D = 2r \cdot |\mathbf{X}| \text{ comparisons.} \tag{394}$$

By way of comparison, computation of the image gradient using a Prewitt operator [82], which requires a three-pixel von Neumann template, incurs work of $3 \cdot |\mathbf{X}|$ addition operations. This represents one of the more efficient (but not necessarily the most effective) methods of edge detection in the image domain. Thus, the efficiency that results from BTC-based edge detection is given by:

$$\eta \geq \frac{W_{\text{ed}}(\mathbf{X})}{W_{\text{BTCed}}(\mathbf{Y}, r)} = \frac{3 \cdot |\mathbf{X}| \text{ additions}}{2r \cdot |\mathbf{X}| \text{ comparisons}} = \frac{1.5}{r}. \tag{395}$$

In edge-intensive imagery (e.g., leaves, woodland scenes, rushing water, views of buildings, or text images) we have found that $0.5 < r < 0.8$ is typical. Thus, in the preceding

equation the efficiency would range from two- to three-fold. However, recall that this is the minimum efficiency achievable, since our cost analysis is based on the Prewitt operator. Given Sobel and Kirsch edge detectors [83,84], which are more burdensome to compute, we have found in practice that tenfold decreases in the computational cost are not unusual with compressive edge detection.

10.2.1.4. Information loss. Since the edge detection operation e discards block means and bits from the zero-crossing bitmap, much information about object structure is lost. Unlike a standard edge image, the image resulting from BTC-based edge detection cannot necessarily be inverted to yield a close approximation to the source image. This is due to (a) possible loss of gradient magnitude in the BTC block representation and (b) the losses inherent in quantizing a greyscale block into a block mean and bit map. Furthermore, e yields only an approximation to the edge detection of the bitmaps, and is thus key to producing an accurate approximation of the resultant edge image. As a result, we are currently researching methods of configuring e to achieve a close approximation to results obtained via image-domain edge detection. Related examples are given in Section 10 for VPIC edge detection and morphological operations.

10.2.2. VPIC-based Edge Detection.

The following description of a VPIC edge detection algorithm is similar to BTC-based edge detection, but requires that the bitmap be retrieved from a codebook. Since the codebook is small, the edge detection operation e can be implemented in a lookup table that accepts codebook indices taken from the nearest neighbors of the compressed pixel of regard. We provide examples from Boolean and greyscale imagery.

10.2.2.1. Algorithm. Let an image $\mathbf{a}$ on $\mathbf{X}$ be compressed by T = VPIC to yield a compressed image $\mathbf{a}_c$ on $\mathbf{Y}$ whose pixel values are of form (μ,g,θ,j), which denote the corresponding source block mean μ, gradient intensity g, gradient orientation θ, and index j of an exemplar in the VPIC codebook $\mathbf{c}$. We assume that the exemplar is rotated through angle θ to yield a best match with the block's zero-crossing bitmap. If a mean block is portrayed by a compressed pixel value $\mathbf{a}_c(\mathbf{y})$, where $\mathbf{y}$ is a point in $\mathbf{Y}$, then the remainder of the parameters in $\mathbf{a}_c(\mathbf{y})$ are zero-valued.

Compressive edge detection using VPIC-compressed imagery involves an edge transform e that has a similar high-level description to the transform e previously employed for BTC-based edge detection. We also employ a rotation transform $rot: \mathbf{R} \times \{0,1\}^{kl} \rightarrow \{0,1\}^{kl}$ that accepts θ and an exemplar $\mathbf{c}(\mathrm{j})$, which is then rotated through angle θ in the output.

Since VPIC quantizes the gradient angle θ into a small number of bins and employs only a few exemplars that are deemed visually significant, the actual codebook size is quite small, rarely exceeding eight exemplars, which can be indexed using three bits. As a result, rather than developing an involved analytical representation of e, we can employ a lookup table g to compute e. For example, let g accept the $\mathbf{y}$-th encoding block and its nearest neighbor blocks, and output the $\mathbf{y}$-th edge detected block. For example, if $\mathbf{X}$ has von Neumann connectivity, then a given block has four neighbors, and $|domain(g)| = (4 + 1)(3) = 15$ bits, or 32K elements. The corresponding LUT has size $15kl$ bits. Thus, if k,l = 4, then each element occupies two bytes, and the LUT size is 64K bytes, which is feasible for some currently-available parallel processors. For Moore connectivity, each block has eight neighbors and the LUT domain has 30 bits = 1G elements, which is prohibitively large. Thus, we prefer von Neumann connectivity for our compressive processing operations.

Compressive edge detection over VPIC-compressed imagery is achieved by performing the following steps:

Step 1. Initialize $\mathbf{b}$, an image on $\mathbf{X}$ to zero.

Step 2. For each $\mathbf{y} \in \mathbf{Y}$, do:

$$\begin{aligned} &\texttt{if } (p_2(\mathbf{a}_c(\mathbf{y})) = 0) \texttt{ then } \mathbf{b}|_{k^*(\mathbf{y})} = 0 \\ &\texttt{else} \\ &\quad \mathbf{b}|_{k^*(\mathbf{y})} = p_2(\mathbf{a}_c(\mathbf{y})) * rot(p_3(\mathbf{a}_c(\mathbf{y})), e[\mathbf{c}(p_4(\mathbf{a}_c(\mathbf{y})))]) \\ &\texttt{endif}. \end{aligned} \tag{396}$$

Apart from block rotation, the treatment of edge and mean blocks is highly similar to BTC-based edge detection.

10.2.2.2. Complexity. Given a total of $|\mathbf{Y}|$ blocks, of which a fraction r are edge blocks, then the preceding VPIC-based edge detection algorithm requires $(r-1) \cdot |\mathbf{Y}| \cdot kI$ I/O operations to set mean blocks to zero. Since edge detection is performed over a small codebook of M exemplars, the cost of M invocations of e is negligible. Thus, the only significant processing overhead is $r \cdot |\mathbf{Y}| \cdot kI$ sign changes (a one-bit operation). Thus, VPIC-based edge detection is very fast, and requires work of only $r \cdot |\mathbf{X}|$ one-bit operations. The resultant efficiency is given by

$$\eta \geq \frac{W_{ed}(\mathbf{X})}{W_{VPICed}(\mathbf{Y}, r)} = \frac{3 \cdot |\mathbf{X}| \text{ additions}}{r \cdot |\mathbf{X}| \text{ comparisons}} = \frac{3 \cdot \Delta t_a}{r}, \tag{397}$$

where Δt_a denotes the processing time (or work) required for addition.

In edge-intensive imagery, we have found that $0.5 < r < 0.8$ is typical. Thus, in the preceding equation, $4 < \eta < 6$. However, recall that this is the minimum efficiency achievable. As opposed to Sobel and Kirsch edge detectors, which are more burdensome to compute, we have found that twenty-fold efficiencies can be realized using VPIC-based compressive edge detection. The edge thickness, which is a primary distinction between various image-domain edge detectors, can be regulated in VPIC by configuring the codebook exemplars to have a thicker edge representation.

10.2.2.3. Example. We present the following example of VPIC edge detector configuration, expressed in terms of the exemplar lookup table. This example holds equally well for greyscale or Boolean imagery. However, one may want to adjust the edge-block mean to account for the intensity difference between the current block (pixel of the compressed image) and its neighbors. Observe that the 4×4-pixel VPIC codebook

$$\mathbf{c}(1)=\begin{bmatrix}1&1&1&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\quad \mathbf{c}(2)=\begin{bmatrix}1&1&1&1\\1&1&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix}$$
$$\mathbf{c}(3)=\begin{bmatrix}1&1&1&1\\1&1&1&0\\1&1&0&0\\1&0&0&0\end{bmatrix}\quad \mathbf{c}(4)=\begin{bmatrix}1&1&0&0\\1&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \tag{398}$$

can be used to represent the following derivation of an edge block

$$\begin{matrix}\ddots & & \\ & \begin{bmatrix}\vdots&\vdots&\vdots&\\1&1&1&1\end{bmatrix} & \\ \begin{bmatrix}&&&1\\ \cdots&&&1\\ \cdots&&&1\\&&&1\end{bmatrix} & \begin{bmatrix}1&1&1&1\\1&1&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix} & \begin{bmatrix}1&\\1&\cdots\\1&\cdots\\1&\end{bmatrix} \\ & \begin{bmatrix}1&1&1&1\\&\vdots&\vdots&\end{bmatrix} & \ddots\end{matrix} \quad\Rightarrow\quad \begin{bmatrix}0&0&0&0\\1&1&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix} \tag{399}$$

in terms of the additional edge representation exemplars

$$\mathbf{c}(5)=\begin{bmatrix}0&0&0&0\\1&1&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix}\quad \mathbf{c}(6)=\begin{bmatrix}0&0&1&1\\0&1&1&0\\1&1&0&0\\1&0&0&0\end{bmatrix}$$
$$\mathbf{c}(7)=\begin{bmatrix}0&1&0&0\\1&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\quad \mathbf{c}(8)=\begin{bmatrix}1&1&1&1\\1&0&0&1\\0&0&0&0\\0&0&0&0\end{bmatrix} \tag{400}$$

and the following derivation:

$$(1|2|3)'''\ \begin{matrix}(1|2|3)''\\2\\1|2|3\end{matrix}\ (1|2|3)' \quad\Rightarrow\quad 5\,, \tag{401}$$

where $(1|2|3)'$ denotes the transpose of exemplar $\mathbf{c}(1)$, $\mathbf{c}(2)$, or $\mathbf{c}(3)$. The preceding derivation represents an efficiency of order CR_D, since one LUT operation could be substituted implementationally for sixteen pixel operations. The following examples are illustrative.

10.2.2.4. Complexity. Given the VPIC codebook index set $\mathbf{U} = \{1,2,...,4\}$, the VPIC Boolean edge detector requires only $|\mathbf{Y}| = 6(7) = 42$ invocations of the lookup table $f: \mathbf{U}^5 \rightarrow \mathbf{U}$. In contrast, a Prewitt-like edge detector that employs a template which is defined by its weights as

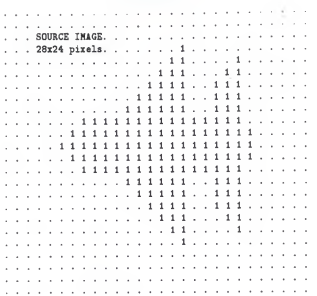
$$t = \begin{pmatrix} & & -1 \\ & & \\ -1 & & 1 \end{pmatrix} \tag{402}$$

requires

$$\begin{aligned} W_{\bigcirc}(|\mathbf{X}|) &= 3 \cdot |\mathbf{X}| = 3 \cdot 28 \cdot 24 \text{ multiplications} \\ &= 2,016 \text{ multiplications}. \end{aligned} \tag{403}$$

Given a ratio r of multiplication delay to lookup table delay, this represents a speedup of $\eta = r \cdot 2016/42 = 48r$, if I/O overhead is neglected. If $r = 6.5$, as is the case for integer multiplication on the Sun Sparcstation-10 on which this algorithm was tested, then $\eta = 312$, which represents a nontrivial speedup that far exceeds CR_D.

10.2.2.5. Example. Let the source image **a** shown in Figure 19a be subjected to an edge detector that preserves the outer boundary, to yield the boundary image of Figure 20a. The VPIC representation of **a** is given in 19b. Such a detector could be implemented in terms of the Prewitt detector [82], or could be realized via the subtraction of a morphologically eroded version of **a** from **a**. Applying the preceding derivational method to VPIC edge detection of **a**, we obtain the image of Figure 20b. Note the encoding errors near the vertices of the airplane's tail and wings. This type of error is typical in low-resolution imagery, and represents the lower limit of VPIC's coding accuracy.



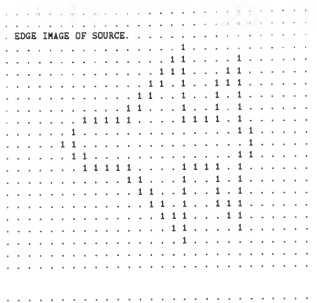
(a)

VPIC Image Column

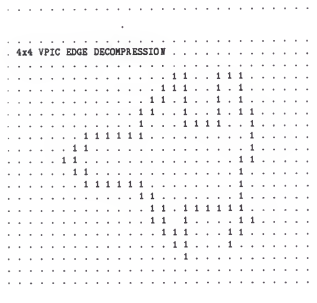
Row	1	2	3	4	5	6	7
1	M,0	M,0	M,0	M,0	M,0	M,0	M,0
2	M,0	M,0	M,0	3'	1'	2'	M,0
3	M,0	4'	-1	M,1	M,1	-1'''	M,0
4	M,0	4''	2	M,1	M,1	2'	M,0
5	M,0	M,0	M,0	4''	1'	4'''	M,0
6	M,0	M,0	M,0	M,0	M,0	M,0	M,0

(b)

Figure 19. Example of VPIC coding of very low-resolution Boolean imagery: (a) source image, (b) VPIC representation of a), where M,x denotes a mean block of mean x.



(c)



(d)

Figure 20. Example of VPIC coding of very low-resolution Boolean imagery: (a) boundary-detection of Figure 19a), (d) VPIC edge detection of Figure 19a).

10.2.2.6. Information loss. We have configured the VPIC edge detector such that the matching of exemplars with the zero-crossing bitmaps incurs less than 25 percent erroneous pixels per block. Given the ratio $N_e/|\mathbf{Y}|$, where N_e denotes the number of edge blocks, we have a total error cross-section of $0.25(N_e/|\mathbf{Y}|)\times 100$ percent of erroneous pixels in the compressed image.

In the preceding example, there were 21 erroneous pixels of $672 = 28\times 24$ pixels, as shown in the following figure. This represents an error rate of $0.3125 = 21/672$, or 3.1 percent probability of error per pixel. From Figure 19b, we see that there are 12 edge blocks, for a total maximum predicted error rate of $0.0714 = (0.25)12/42$. Assuming that the mean rate is half the maximum, we have a mean predicted error rate of 0.0357, or 3.6 percent of source pixels. The measured error rate of 3.1 percent of source pixels varies from the predicted rate by 3.36 pixels = (0.035–0.031)(24×28pixels), or 0.5 percent of the source domain size. In practice, we find that the predicted mean error rate consistently approximates the actual rate to within 10 percent of the image size when a 25 percent error cross-section is assumed in the VPIC match.

	Column of VPIC Representation						
Row	1	2	3	4	5	6	7
1	-	-	-	1	-	-	-
2	-	-	-	-	-	1	-
3	-	-	1	-	2	2	-
4	-	3	1	-	4	1	-
5	-	-	-	-	1	3	-
6	-	-	-	-	-	-	-

Figure 21. Error analysis of the edge detector in Figure 20b, in terms of erroneous source pixels per encoding block.

10.2.2.7. Example. Greylevel VPIC edge detection is illustrated in the following figure, in which a noisy, slightly blurred 720×480-pixel underwater (UW) image was subjected to Sobel edge detection as well as VPIC edge detection. An error image was obtained by differencing the Sobel and VPIC edge detectors. Note that the error graph contrasts the UW case with terrestrial imagery of small targets against a grassy background. In both cases, error increases as small features are obliterated. However, in the land-based imagery, which has much lower spatial frequency content over the periods ranging from five to eight pixels, the combined camera noise and VPIC representational error did not exceed 3.2 bpp for blocksizes as large as 8 pixels square.

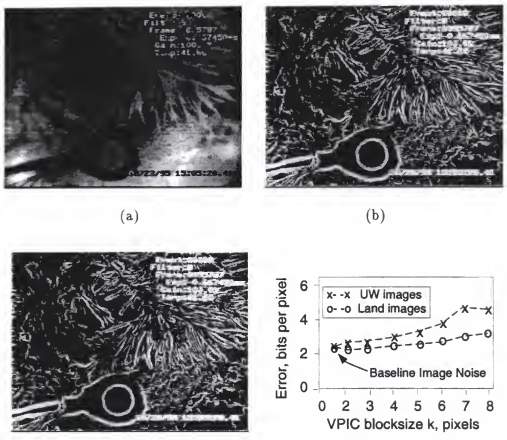


Figure 22. Greylevel edge detection with VPIC: (a) source image, (b) Sobel edge detection, (c) 4×4–pixel VPIC edge detection using the codebook similar to that given in Example 10.2.2.3, (d) noise and representational error as a function of VPIC blocksize (k×k pixels) for underwater and land-based imagery.

10.2.3. VQ-based Edge Detection.

We begin by noting several useful properties of the VQ exemplars.

10.2.3.1. Observation. Cosman et al. [1,12] have shown that VQ-compressed imagery can be thresholded according to a minimum-entropy criterion to achieve a very approximate method of edge detection. Unfortunately, certain exemplars that have no edges are frequently juxtaposed with those that contain high-intensity edge information (i.e., high variance). The resulting "blocking effect" at block boundaries can lead to visually-apparent noise in the edge image if the blocksize is large in relation to spatial features of interest. As discussed previously, the edge detection which is local to a given block produces boundary effects that can be mistaken for linear features.

10.2.3.2. Algorithm. As an improvement on Cosman's method, we have modified the method employed in VQ smoothing to facilitate edge detection by applying a given edge detector to the exemplars in the VQ codebook **c**. Given a codebook of M exemplars, this requires $\mathbf{O}(M)$ effort and $\mathbf{O}(Mkl)$ pixels are processed. Upon image reconstruction, the exemplars are substituted into the reconstructed image, and an approximation to the edge-detected source image results. Blocking effects can be reduced to a visually acceptable level by extending the boundaries of each codebook exemplar by replicating m row or column pixels outward using randomly chosen values, where m denotes the halfwidth of the edge detector's sampling window. The block extensions are discarded after edge detection occurs, thus reducing boundary effects. Since the VQ codebook is specific to a given image training set, this method is more useful than a specific instance of BTC- or JPEG-based edge detection. However, VQ based edge detection is particular to a given training set, as opposed to VPIC-based filtering, which is based on a data-independent codebook.

10.3. VPIC Morphological Operations.

VPIC morphological operations are similar implementationally to VPIC edge detection, and exhibit corresponding advantages of speed and simplicity. For purposes of brevity, we consider Boolean erosion and dilation with the von Neumann template.

10.3.1. Concept.

We begin with a brief overview of erosion and dilation.

10.3.1.1. Observation. Ritter [54] has shown that morphological erosion and dilation with the von Neumann template can be implemented in terms of the image algebra additive maximum and minimum. For example, given finite $\mathbf{X} \subset \mathbb{Z}^2$, the erosion of a Boolean source image $\mathbf{a} \in [0,1]^{\mathbf{X}}$ by a von Neumann template $\mathbf{t} = \begin{pmatrix} & 1 & \\ 1 & I & 1 \\ & 1 & \end{pmatrix}$ is expressed as $\hat{\mathbf{a}} = \mathbf{a} \textcircled{\wedge} \mathbf{t}$.

10.3.1.2. Example. The erosion of a source image

$$\mathbf{a} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{404}$$

by $\mathbf{t} = \begin{pmatrix} & 1 & \\ 1 & I & 1 \\ & 1 & \end{pmatrix}$ is given by

$$\hat{\mathbf{a}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{405}$$

10.3.1.3. Remark. Whereas erosion removes pixels from a unitarily-valued boundary, dilation adds pixels to the boundary. In the preceding example, the dilation of $\hat{\mathbf{a}}$ (Equation 405) by $\mathbf{t}$ yields the image $\mathbf{a}$ (Equation 404). We thus write $\mathbf{a} = \hat{\mathbf{a}} \textcircled{\vee} \mathbf{t}$.

10.3.1.4. Complexity. If multiplications are employed for combination of an image with $\mathbf{t}$ using $\bigwedge$ or $\bigvee$, then a total of five multiplications and four maximum operations are required per pixel in $\mathbf{X}$. Otherwise, a lookup table may be employed that has five-bit input and one-bit output, for a total space requirement of 32 bits and a total cost of $|\mathbf{X}|$ LUT invocations and $6 \cdot |\mathbf{X}|$ I/O operations.

Since there is no information loss in Boolean erosion/dilation, we proceed to a discussion of VPIC erosion.

10.3.2. VPIC Erosion or Dilation.

VPIC-based erosion has the advantage of requiring I/O operations only, if the implementation uses a lookup table similar to that required for VPIC edge detection.

10.3.2.1. Observation. Recall that VPIC represents an edge block of a Boolean image (i.e., a zero-crossing map that has both ones and zeroes) in terms of zero block mean and standard deviation, with a given exemplar index. Thus, we can derive a codebook specific to morphological erosion, similar to the codebook we derived for edge detection. The following example is illustrative.

10.3.2.2. Example. Observe that the VPIC codebook $\mathbf{c}$ given in Example 10.2.2.3 can be used to express the following derivation of an eroded block exemplar

$$\begin{matrix} \ddots & & \begin{bmatrix} \vdots & \vdots & \\ 1 & 1 & 1 & 1 \end{bmatrix} \\ \begin{bmatrix} & 1 \\ \cdots & 1 \\ \cdots & 1 \\ & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ \vdots & \vdots & & \end{bmatrix} & \begin{bmatrix} 1 & \\ 1 & \cdots \\ 1 & \cdots \\ 1 & \\ & \ddots \end{bmatrix} \end{matrix} \quad \Rightarrow \quad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{406}$$

as

$$(1|2|3)''' \quad \begin{matrix} (1|2|3)'' \\ 2 \\ 1|2|3 \end{matrix} \quad (1|2|3)' \quad \Rightarrow \quad 8 \,, \tag{407}$$

where $(1|2|3)'$ was defined in Section 10.2.2.3.

10.3.2.3. Example. Similarly, **c** can be used to express the following dilation of a VPIC block exemplar

$$\begin{matrix} \ddots & & \\ & 1 & \\ \cdots & 1 & \\ \cdots & 1 & \\ & 1 & \end{matrix} \quad \begin{matrix} \begin{bmatrix} \vdots & \vdots & \vdots & \\ 1 & 1 & 1 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \end{bmatrix} \end{matrix} \quad \begin{matrix} \begin{bmatrix} 1 & \\ 1 & \cdots \\ 1 & \cdots \\ 1 & \end{bmatrix} \\ \ddots \end{matrix} \quad \Rightarrow \quad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{408}$$

as

$$(1|2|3)''' \quad \begin{matrix} (1|2|3)'' \\ 2 \\ 1|2|3 \end{matrix} \quad (1|2|3)' \quad \Rightarrow \quad -1'', \tag{409}$$

where $-1''$ denotes the Boolean negation of $\mathbf{c}(1)$ transposed twice. Note that the Hamming distance between the result of Equation 408 and the exemplar $\neg(\mathbf{c}(1))''$ is two, which is within the error limit of 25 percent of total pixels discussed in Section 10.2.2.6.

10.3.2.4. Remark. The discussion of complexity and information loss given in Section 10.2.2 holds for VPIC morphological operations as well. Preliminary tests on Boolean text imagery indicate that VPIC dilation incurs greater error than VPIC erosion. This is a reasonable observation, since dilation fills the von Neumann neighborhood of a pixel with unitary values, while erosion may remove one or more unitary values from the neighborhood of a given pixel. That is, erosion and dilation are not inverses of each other, and should be expected to exhibit differing error figures.

10.3.2.5. Computational efficiency. VPIC morphological operations that employ the von Neumann template require six I/O operations (five inputs, one output) and one LUT invocation. Assuming that a VPIC erosion/dilation LUT invocation incurs identical overhead to that incurred by an erosion/dilation LUT over source imagery (per Section 10.3.1.4), the cost per pixel of the preceding method of VPIC erosion would be identical to the cost of noncompressive erosion. A symmetric observation holds for dilation.

However, VPIC morphological operations with the von Neumann template affect $k\ell$ pixels (the encoding block size), versus one pixel with image erosion. Thus, the resultant efficiency $\eta = CR_D$.

We are currently investigating the configuration of VPIC codebooks for multi-pixel erosion with von Neumann and non-von Neumann templates. For example, in the absence of boundary information, two-pixel erosion would convert $\neg\mathbf{c}(1)$ to $\mathbf{c}(1)$. An additional topic of interest is the description of VPIC block formation rules (per Equation 409) in terms of a grammatical representation.

10.4. High-Level Compressive Computations.

Practical image processing of multi-target fields can benefit from efficient methods of target classification, as well as methods of identifying target shapes. In this section, we present an algorithm for target classification based on BTC- or VPIC-compressed imagery (Section 10.4.1) and VPIC-based connected component labelling (Section 10.4.2).

10.4.1. Target classification.

We begin by observing that targets can be classified by their statistical measures, which are encoded in VPIC and BTC imagery. This observation is elaborated to include detection of pixel-level target features.

10.4.1.1. Observation. Naturally-occurring objects or groups of objects tend to be irregularly shaped and have relatively rough textures (e.g., rocks, a field of grass, or tree bark). In contrast, manufactured objects are generally regularly shaped and tend to have smooth surfaces due to the requirements of drawing, forming, and stamping processes. It is well known that, given a multispectral image, object color can be roughly assessed using local averages across taken across image partitions in each band whose size approximates the expected size of one or more target(s) of interest. We have elsewhere noted [62,81]

that surface roughness or texture can be approximated by the image gradient or standard deviation. As a result, manufactured objects can be detected in natural scenes by first selecting candidate image areas according to a local mean criterion, then thresholding the candidate neighborhoods based on local variance criteria. Since mean and variance information are present in BTC- and VPIC-compressed imagery, it follows that, by thresholding such imagery as previously mentioned, one could classify targets of interest.

10.4.1.2. Algorithm. Let a multi-target image $\mathbf{a} \in F^{\mathbf{X}}$ be compressed by a transform T = BTC to yield an image $\mathbf{a}_c \in G^{\mathbf{Y}}$. Recall that each $k \times l$-pixel encoding block $\mathbf{b}$ in $\mathbf{a}$ is represented by the block mean μ, standard deviation σ, and zero-crossing bitmap $\mathbf{d} \in \{0,1\}^{kl}$.

Given probable target intensities in $T \subset range(\mathbf{a})$, as well as probable target textures represented by standard deviations in V, the set of probable target locations $D \subset \mathbf{X}$ is given by:

$$D = h^*[domain(\chi_{\in T}(p_1[\mathbf{a}_c(\mathbf{Y})]) * \chi_{\in V}(p_2[\mathbf{a}_c(\mathbf{Y})]))], \tag{410}$$

where the indexing function h^* was defined previously. In a more efficient linear combination scheme, we could substitute addition for Hadamard multiplication, denoted by (*) in the preceding equation.

10.4.1.3. Complexity. If T and V are each specified in terms of an interval on $range(\mathbf{a})$, then the preceding algorithm requires $4 \cdot |\mathbf{Y}|$ comparisons and $3 \cdot |\mathbf{Y}|$ combination operations, which can be implemented via an addition or logical *and* operation. As a result, processing requires $\mathbf{O}(|\mathbf{Y}|)$ time. By way of comparison, signature determination over $\mathbf{X}$ requires $2 \cdot |\mathbf{X}|$ comparisons per spectral band. The computational efficiency of BTC-based compressive target recognition is clearly $\mathbf{O}(CR_D)$, and usually exceeds CR_D due to the burdensome operations required for local averaging and computation of the standard deviation over $\mathbf{X}$.

10.4.1.4. Observation. In the case of VPIC-compressed imagery, a target is characterized by a variance measure that is derived from the encoding block gradient intensity. One thresholds the VPIC image according to mean and variance in order to obtain candidate target points in the compressed domain |Y|. Thus, the VPIC gradient orientation and exemplar index can be ignored in this simple formulation. Since the mean and an approximation to standard deviation is given, one can implement filters such as the Double-Gated Filter [81] using a reduced template representation. The problem of implementing ATR filters over VPIC-encoded imagery will be addressed in a future publication.

10.4.1.5. Algorithm. In more involved implementations of target classification, we employ the VPIC exemplars, as follows:

Step 1. Given target intensities in T and variance parameters in V, compute Equation 410 to yield probable target locations in D.

Step 2. Compute C = h(D), which are the target locations in **Y**, where the indexing function h was defined previously.

Step 3. Assuming that target size and shape is known, arrange the VPIC codebook exemplars to yield an approximate representation of target shape, as follows:

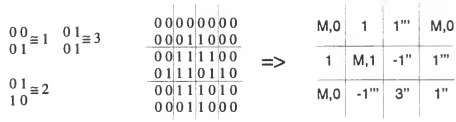


Figure 23. Target characterization using VPIC codebook exemplars with indices in {1,2,3}: (a) VPIC codebook, (b) source image with unitarily-valued target region, (c) compressed representation, where 1′ denotes exemplar 1 rotated by 90 degrees.

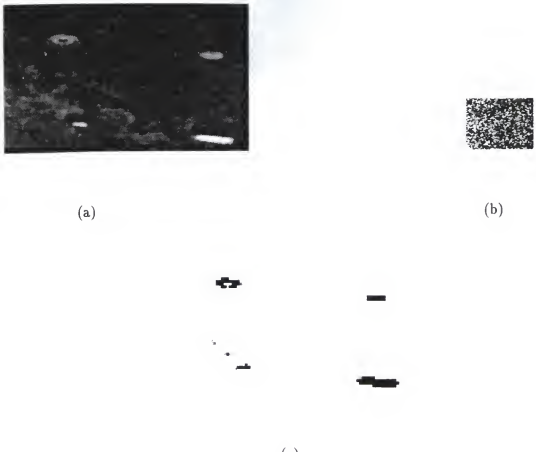
Step 4. Perform Step 3 for blocks whose indices are in C by matching VPIC exemplars to the combined threshold results (e.g., the product in Equation 410) restricted to h^*(C).

Step 5. Compare the results of Step 3 for each contiguous group of blocks found in Step 2 to yield a measure of target presence.

The algorithm operates as follows. Steps 1 and 2 compute indices in C of candidate target blocks in **a**. Step 3 produces a probable target template. Step 4 correlates the target template developed in Step 3 with the VPIC compressed image's exemplar indices to determine target presence. Additionally, the known data about target mean and standard deviation are used to restrict the source image to candidate target locations. When these two partial results are combined (usually via addition and thresholding), one obtains an image of candidate target locations where the block mean, gradient intensity, orientation, and exemplar indices are specific to blocks whose indices are in C. If a match is found, then target presence is indicated. At this point, one can check the uncompressed source image to achieve greater accuracy in target discrimination.

10.4.1.6. Complexity. Steps 1 and 2 of Algorithm 10.4.1.5 require one computation of Equation 410 and |D| applications of h^*, respectively. Step 3 depends upon target size and blocksize used in VPIC-compressing the target representation shown in Figure 24b. Step 4 can be obtained with no significant effort by restricting the compressed image to points in C. Step 5, which requires at most $\mathbf{O}(|C|^2)$ comparisons, is the costliest step.

10.4.1.7. Example. Figure 24a shows a multi-target image that contains four targets of varying sizes and visibility, all of which are partially resolved (5 to 100 pixels in size). Figure 24b shows the compressed target field image that resulted from VPIC applied to (a), and Figure 24c illustrates the output of Equation 410 following decompression. All four targets are correctly detected, although there are several false target pixels above the bottom left-hand target, which are due to misidentification of noise pixels in that area and can be removed by morphological erosion and dilation.



(a) (b)

(c)

Figure 24. Example of VPIC-based target recognition: (a) source image, (b) BTC-compressed image over a portion of which the target recognition algorithm (Equation 410) computes, (c) decompressed target location image.

10.4.2. VPIC Connected Component Labelling.

We conclude our presentation of compressive processing with an overview of a more difficult problem, namely connected component labelling (CCL). Here, we threshold an image such as that shown in Figure 24c and label each contiguous unitarily-valued region with a unique number. For purposes of simplicity, we do not require sequential CCL, which can nevertheless be implemented by relabelling the result of VPIC CCL.

10.4.2.1. Observation. VPIC encodes a Boolean image by assuming that encoding block contents are zero-crossing bitmaps of encoding blocks that have zero mean and unit gradient intensity. Each block is represented in the VPIC image by a best-match exemplar from a small codebook (typically less than eight exemplars). Since the codebook exemplars are Boolean images that are known a priori, we can tag each codebook exemplar with a tag vector (N,E,W,S), where N = 1 if the North (top) row of a given exemplar is unitarily valued or has a majority of one-valued pixels, and N = 0 otherwise. Symmetric operations are applied to the E, W, and S boundaries. This facilitates the determination of adjacency between blocks, which greatly simplifies CCL.

10.4.2.2. Assumption. Assume that an image contains the values of encoding blocks in a source image $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, which is obtained by applying a transform T_{VPIC} to $\mathbf{a}$, thus obtaining a compressed image $\mathbf{a}_c \in \mathbf{G}^{\mathbf{Y}}$. Further assume that we perform connected component labelling to yield an image $\mathbf{c} = \mathrm{CCL}(\mathbf{a})$.

Let $\mathbf{Y}$ be two-dimensional, and let a compressed block representation whose domain point $\mathbf{y} \in \mathbf{Y}$ is located in the row and column directly beneath a block whose domain point is denoted by $\mathbf{z}$. If the N-tag of the exemplar indexed by $p_4(\mathbf{a}_c(\mathbf{y}))$ and the S-tag of the exemplar indexed by $p_4(\mathbf{a}_c(\mathbf{z}))$ are both unitarily-valued, then we say the blocks are *adjacent*. Symmetric cases hold for the E, W, and S tags.

From such interrogation, which requires a maximum of $\mathbf{O}(|\mathbf{Y}|)$ time, where $\mathbf{Y}$ denotes the compressed domain, an adjacency graph

$$G = (\mathrm{V}, \mathrm{E}) \in (\mathbf{Y}, \mathbf{Y} \times \mathbf{Y}) \tag{411}$$

is constructed, whose edge set $\mathrm{E} \subset \mathbf{Y} \times \mathbf{Y}$. This process is possible using $\mathbf{a}_c$ only, since we assume that the block means which represent targets are greater than zero.

10.4:2.3. Algorithm. Given the adjacency graph G, we next assign unique positive integers to pixels in an image $\mathbf{e}$ on $\mathbf{Y}$. The image $\mathbf{f}$ that contains the indices of connected components on $\mathbf{Y}$ is computed as follows:

Step 1. For each point $\mathbf{y} \in \mathbf{Y}$, interrogate the points $\mathbf{z} \in N(\mathbf{y})$, where N denotes the von Neumann neighborhood of $\mathbf{y}$, such that if $\mathrm{E}(\mathbf{y},\mathbf{z})$ exists in G, then

$$\mathbf{e}(\mathbf{y}) = \vee\,(\mathbf{e}(\mathbf{y}), \mathbf{e}(\mathbf{z})). \tag{412}$$

Step 2. Apply the zero-valued, two-dimensional von Neumann template $\mathbf{t}$ to $\mathbf{e}$ as follows:

$$\begin{aligned} &\mathbf{m} = \chi_{>0}(p_1(\mathbf{a}_c)) \\ &\texttt{repeat}: \\ &\{\,\mathbf{f} = \mathbf{m} * (\mathbf{e} \boxtimes \mathbf{t}) \\ &\quad \texttt{if } \mathbf{f} = \mathbf{e} \texttt{ then stop}, \texttt{ else } \mathbf{e} = \mathbf{f} \texttt{ endif}\,\}. \end{aligned} \tag{413}$$

The algorithm operates as follows. Step 1 ensures that a large value in $\mathbf{e}$ is placed on a given component in $\mathbf{a}_c$, thereby facilitating the subsequent maximum operation. Step 2 propagates the maximum value of each component throughout the component to yield an image $\mathbf{f}$ of uniquely labelled components.

10.4.2.4. Complexity. In practice, given a compressed image $\mathbf{a}_c$, the formation of G requires $4 \cdot |\mathbf{Y}|$ comparisons. Given convex components of maximum radius r, Ritter [7] has shown that the labelling algorithm of Step 2 requires at least r/2 iterations of the inner loop, which equates to $2.5\mathrm{r} \cdot |\mathbf{Y}|$ additions and $2\mathrm{r} \cdot |\mathbf{Y}|$ maximum operations, since the von Neumann template has five pixels. Assuming that the addition, maximum, and comparison operations require identical work, then the computational efficiency of CCL

over VPIC-compressed imagery, as opposed to applying Equation 413 to uncompressed imagery containing convex hulls, is given by:

$$\eta = \frac{W_{CCL}(\mathbf{X})}{W_{VCCL}(\mathbf{Y})} = \frac{4.5 \cdot |\mathbf{X}| \text{ operations}}{8.5 \cdot |\mathbf{Y}| \text{ operations}} = 0.53 \cdot CR_D \,. \tag{414}$$

If $k,l = 4$ (which equates to $CR_D = 16$), then $\eta = 8.47$, under constraint of the preceding assumptions. Therefore, compressive processing with $\mathbf{O}(CR_D)$ efficiency is feasible.

CHAPTER 11
APPLICATIONS IN PARALLEL COMPUTING

In this chapter, we consider applications of the theory developed previous chapters, particularly concerning implementation on SIMD-parallel meshes. We begin with a brief discussion of the effect of domain compression ratio (Section 11.1), then progress to range compression ratio effects (Section 11.2). We conclude with a discussion of mapping image operations to more efficient operations over the corresponding compressed format (Section 11.3) and image partitioning efficiency in compressive processing (Section 11.4).

11.1. Effect of Domain Compression Ratio.

We have previously shown that the domain compression ratio CR_D can be supra-unitary, which indicates a reduced number of pixels in the compressed image representation versus the source domain. For example, if $k \times l$-pixel encoding blocks are employed in VPIC, then each VPIC exemplar referenced in a given compressed pixel is associated with kl source pixels. This seemingly obvious fact leads to the following three important consequences for parallel implementation of compressive operations:

1. *The number of processors required to compute an image operation can be reduced by* $\mathbf{O}(CR_D)$. However, such reduction is achieved in the absence of computational speedup due to fewer pixels. In practice, reduced parallelism is achieved when the number of compressed pixels per processing element (PE) is greater than or equal to the number of pixels per PE for the uncompressed image. Any additional speedup derives from a reduced word width in the compressed pixel, if such effects are present (reference Section 11.2). In this chapter, we assume that one pixel

is stored per PE, with nearest-neighbor pixel values input and stored on-board a given PE in registers, as required.

2. *I/O requirements can be correspondingly decreased for compressed imagery.* This is a direct consequence of the reduced data burden incurred by $CR_D > 1$. If the source imagery is tesselated blockwise (e.g., $k \times l$ pixel rectangular blocks) and an $n \times m$-processor SIMD mesh with n-channel parallel I/O is required for an image operation, then it is possible that the mesh size can be reduced to n/k by m/l PEs, with only n/k-channel parallel I/O required. Alternatively, the n-fold I/O parallelism can be employed to speed up I/O by a factor of k, using input of multiple image rows. The rows would then be shifted across the mesh to their appropriate positions, which would require additional I/O overhead. Depending upon n,m,k,*l*, and the requirements of computations that can be pipelined with of follow the I/O operations, this additional overhead may or may not be burdensome.
3. *If the topology of the compressed image is isomorphic to (a) the topology required by a given compressive operation and (b) the inter-processor connection scheme, then near-optimal mapping of the compressive operation to the target architecture is possible.* A case in point is VPIC edge detection with the von Neumann topology, square encoding blocks, and a von Neumann template or subset thereof. Recall that we have shown (in Section 10.2.2) that nearest-neighbor (NN) data dependencies are required for VPIC edge detection. For example, consider image-template operations with the von Neumann template. Assuming von Neumann source and compressed domain topologies that are isomorphic to the processor mesh connectivity, we have isomorphism between the source, compressed, and mesh domains. Thus, we can map an algorithm with NN communication in the image domain to an algorithm with NN communication requirements in the compressed domain, which can then be implemented on a processor with native NN communication. This is clearly an optimal situation for I/O efficiency, *provided*

that additional I/O overhead is not required by the compressive operation beyond that needed for the noncompressive operation. In other words, we do not want the VPIC edge detector to require NN communication in the Moore sense, versus the von Neumann connectivity supported in hardware on many 2–D meshes. That is, Moore NN communication would require two shifting operations per pixel to read pixel values in diagonal NNs.

As an example of Item 2, above, consider the following simple example. Let a 4×4–processor mesh be employed for image-domain operations. Assume that a 4×4–pixel source image (having 16 pixels) can be compressed at $CR_D = 4$ to yield a 2×2–pixel image, as shown in Figure 25. Further assume that the image and compressive operations fulfill the criteria of Item 3, above. Since only 25 percent of the mesh is now required for image processing, we can construct a pipelined SIMD mesh (similar to a systolic processor) using no additional hardware other than the n parallel I/O channels and the n^2 PEs previously available. Considering the effect of pipeline setup, the throughput is increased nearly four times for noncompressive processing. If operations 1–4 performed in each quadrant of the mesh require similar computational delay (e.g., are implemented using lookup tables, similar to VPIC edge detection and morphological operations), then near-optimal throughput could be achieved.

The potential impact of the preceding concepts on the design of image processing circuitry could be significant. For example, we currently load an image onto a SIMD mesh, process the image, then (frequently) store the intermediate result to disk. This requires considerable I/O overhead. In contrast, assume that we can consistently reduce imagery to an easily-manipulated compressive format (e.g., VPIC or VQ) where compressive operations have template configurations or inter-pixel communication requirements that can be efficiently accommodated by the processor mesh. Further assume that local memories will increase in capacity such that LUT-based processing (similar to VPIC edge detection

or CCL) can be accommodated on-board with no additional I/O overhead due to LUT loading/unloading. (This is a reasonable assumption, given current developments in memory technology).

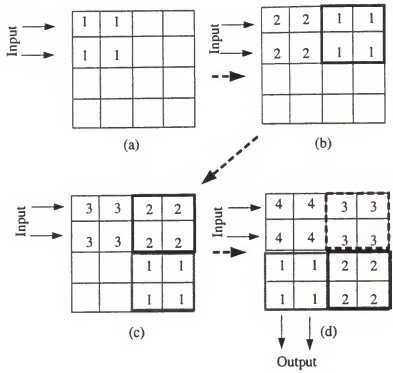


Figure 25. Conversion of a SIMD-parallel mesh to pipelined computation using compressive processing: (a) input and compute over compressed image 1, (b) input compressed image 2 and compute over image 1, (c) input compressed image 3 and compute over images 1 and 2, (d) input compressed image 4, compute over compressed images 1–3, and output compressed image 1.

Now, suppose that we plan to perform production image processing in a manner similar to that shown in Figure 25. If a given architecture could be optimally or near-optimally configured by allocating mesh partitions for various algorithm steps in a manner that balanced image partitioning efficiency with pipeline throughput, then this would represent a significant advance in SIMD-parallel processing efficiency. In particular, intermediate results could be stored on (and circulated around) the various mesh partitions until output to disk storage. Additionally, if the pipeline partitions were allocated correctly, intermediate results could be migrated to other mesh partitions during processing, thereby obviating the

need for I/O of some intermediate results from/to external storage (a time-consuming process).

11.2. Effect of Range Compression Ratio.

A second effect of interest to parallel computing is the compression of source image values can be compressed to yield fewer bits per pixel. For example, consider BTC or VPIC, where a k×k block of m-bit pixels (k^2m bits) is reduced to n bits (e.g., a block mean and variance measure), where n<m is typical. We have shown that arithmetic operations over the block mean and standard deviation are feasible. This means that not only do we achieve efficiency of $\mathbf{O}(CR_D)$ by processing fewer pixels, but we also have fewer bits to process in a given compressive pixel, versus the bits required by a source pixel.

The implications of such range compression effects upon the design of compressive processing hardware are significant. First, if an operationally feasible number of image processing operations can be cascaded without incurring prohibitive error, then it would be possible to construct smaller pipelines (due to a reduced word width) that could perform the compressive operations more efficiently *per compressed pixel value*. When this effect is compounded with the space efficiencies gained by domain compression and pipelining via mesh partitioning (per Figure 25), processing efficiencies that approach or exceed the compression ratio $CR = CR_D \cdot CR_R$ are feasible.

An additional advantage of the range compression effect would be the use of smaller PEs in real-time image processors (e.g., ATR hardware operating at high frame rates under stringent space constraints). For example, if we can compress k^2 m-bit pixels to an n < m bit mean and standard deviation, then target classification may be conducted directly on the compressed image, per Section 10.3.2. Whether this is conducted on board or at a remote processing site (e.g., a supervisory processor on an autonomous sensor platform), the advantages are apparent: (a) smaller size, reduced power consumption and weight, (b)

potentially faster computation due to the compounding of domain and range compression effects, and (c) possible noise averaging and estimation due to the computation of block mean and variance measures, respectfully. The latter advantage is especially important when ATR filter performance depends upon the image noise figure [62,81].

11.3. Simplification of Operations.

As a final note, we summarize several issues pertaining to the replacement of burdensome operations with more efficient operations.

11.3.1. Optimizing the Cost of an Operation.

Recall the case of VPIC edge detection (Section 10.2), where we replaced a costly n<6 pixel convolution over $|\mathbf{X}|$ source pixels with a simple lookup table that computed over $|\mathbf{Y}| \ll |\mathbf{X}|$ pixels. The convolution required at least two multiplications and one addition, whereas the LUT required only six I/O operations (five inputs and one output). The following table illustrates the cost of SIMD-parallel computation of such an operation.

Table 1. Costs involved in SIMD-parallel computation of a 2–pixel image-template convolution versus a 5–pixel LUT operation over VPIC-format imagery.

Algorithm Step (Processor Cycle)	Uncompressed	Compressive
1. Load image&template values (LUT inputs)	2n inputs	5 inputs
2. Multiply values (apply LUT)	n multiplies	1 LUT access
3. Combine partial products	n-1 additions	—
4. Output result	1 output	1 output

11.3.1.1. I/O cost. Assuming that all I/O operations are the same, we have that an n<6 pixel von Neumann template convolution requires 2n+1 I/O operations per pixel in $\mathbf{X}$, versus 6 I/O operations and one memory operation per pixel in $\mathbf{Y}$ for the compressive operation. Inter-processor communication is nearest-neighbor only, in the von Neuman sense, for both operations.

11.3.1.2. Computational cost. The uncompressive operation requires n multiplies and n-1 additions per pixel in **X**, while the compressive operation requires only one LUT access (usually faster than an inter-processor I/O operation).

Thus, if we assume that the compressive operation requires seven I/O operations (one for the LUT invocation and six for values), then the uncompressive operation will match this I/O overhead when n=3. In addition, the uncompressive operation requires multiplications and additions, which are drastically slower than I/O operations. As a result, it is reasonable to have as a future research goal the implementation of compressive image operations in terms of lookup tables. If such techniques are feasible for SIMD-parallel processors, this strategy (coupled with the reduction in I/O cost achievable through reduced parallelism) could yield drastic increases in image processing throughput.

11.3.1.3. Remark. However, the preceding analysis assumes that all multiplications and additions can be completed in one major processor cycle. This is certainly not the case with a synchronous SIMD mesh. We thus revise Table 1 to yield the following cost budget.

Table 2. Processor cycles incurred by SIMD-parallel computation of a 2–pixel image-template convolution versus a 5–pixel LUT operation over VPIC-format imagery.

Algorithm Step	Uncompressed	Compressive
1. Load image&template values (LUT inputs)	2n cycles	5 cycles
2. Multiply values (apply LUT)	$n \cdot c_x$ cycles	1 cycle
3. Combine partial products	$(n-1) \cdot c_+$ cycles	—
4. Output result	1 cycle	1 cycle
Total Cycles Predicted	$n(c_x+c_+)+3) - c_+5+1$	

For purposes of illustration, assume that one multiplication requires 20 cycles and an addition requires 8 cycles, we would have 30n - 7 cycles for one invocation of the non-compressive operation and seven cycles for the compressive operation. Such comparisons provide a clear illustration of the advantages of compressive processing in situations where

isomorphism exists between the topological/connectivity requirements of the source image tesselation, image operation, compressive operation, and mesh.

11.4. Partitioning Efficiency.

We have noted elsewhere [6] that compressive processing facilitates the restructuring of individual operations, by modifying the connectivity of dataflow graphs of certain parallel algorithms such as connected component labelling. An additional advantage of compressive processing that is important for SIMD-parallel computation is the increase in efficiency with which an image can be partitioned

11.4.1. Basic Concepts.

The optimal partitioning of imagery is a key challenge in parallel image processing, due (for example) to the I/O overhead required to swap image partitions on SIMD-parallel arrays.

11.4.1.1. Observation. Given a parallel processor such as a SIMD-parallel mesh with N processors, consider the partitioning of $|\mathbf{X}|$ source data to be optimally allocated to the N-processor mesh. For example, in pointwise operations, we could divide the source data into $\lceil |\mathbf{X}|/N \rceil$ partitions, each of which would be loaded and processed independently. If global-reduce operations are employed, then a global-reduce result from the i-th partition would be stored as intermediate data and would be accumulated with the subsequent (i+1-th) partition's global reduce result.

11.4.1.2. Remark. Image partitioning for pointwise and global reduce operations is relatively straightforward, and generally requires only non-overlapping partitions. However, when image-template operations are employed, template boundary conditions require overlapping partitions. For example, given a $k \times l$-pixel template with rectangular support, a

two-dimensional $|\mathbf{X}|$-pixel source image would require partitions that had overlap of $\lfloor k/2 \rfloor$ pixels at the top and bottom of the partition and $\lfloor l/2 \rfloor$ pixels at the sides of the partition. As a result, for odd-dimensioned rectangular templates, the partition size would be increased by k-1 pixels in the vertical direction and l-1 in the horizontal direction. Since only N processors are available, this implies an increase in the number of partitions. Consequently, the number of processing stages can increase, thereby decreasing throughput. This effect is pronounced for large or non-rectangular templates, and can significantly degrade performance on small meshes, due to the repeated loading of boundary pixels.

11.4.1.3. Observation. An additional problem in image-template operations is the storage of boundary conditions for recursive neighborhood operations. Although we do not extensively discuss recursive image algebra in this study, we note that recursive image-template operations have the additional requirement that partitions be processed sequentially. In such cases, out-of-order execution may be less applicable as a means of achieving near-optimal functional parallelism. The impact upon data parallelism is twofold: (1) in-order partitioning is required, and (2) intermediate results must be stored in boundary pixels (or PEs) on the mesh or registers associated with the mesh. The requirement of within-PE storage can further decrease effective partition size and can thus increase algorithm throughput by decreasing costs associated with I/O and partition swapping costs.

11.4.2. Partitioning Efficiency in Compressive Processing.

The advantage of compressive processing in reducing partition swapping overhead results primarily from block compression, where source pixels are grouped into rectangular blocks that are characterized by few parameters.

11.4.2.1. Observation. Given a supra-unitary domain compression ratio CR_D and a block tesselation scheme that is indexed isomorphically to the image domain coordinate system, it is possible that pointwise and global-reduce operations can be partitioned such that $\lceil |\mathbf{Y}|/N \rceil < \lceil |\mathbf{X}|/N \rceil$ partitions are required, where $\mathbf{X}$ and $\mathbf{Y}$ denote the source and compressed domains, respectively. Further decreases in parallelism can be effected by codebook processing in block-encoded imagery, versus processing the $|\mathbf{Y}|$ encoding blocks.

11.4.2.2. Example. Consider VQ-compressed imagery, where M $k \times l$-pixel exemplars in codebook $\mathbf{c}$ represent encoding blocks directly (i.e., are not scaled or normalized). When computing a blockwise operation over the codebook, a total space requirement of Mkl pixels is present, instead of $|\mathbf{Y}| > Mkl$ or $|\mathbf{X}| > |\mathbf{Y}|$ pixels.

11.4.2.3. Example. Consider the blockwise operation of image addition. Instead of computing $|\mathbf{X}|$ operations in the image domain, we can approximate the effect of VQ codebook addition by adding the exemplars $\mathbf{d}(i,j) = \mathbf{c}(i) + \mathbf{c}(j)$, for all codebook indices $1 \leq i,j \leq M$. By eliminating the lower triangular portion of *domain*($\mathbf{d}$), the new codebook $\mathbf{d}$ will have $(M^2 + M)/2$ exemplars. For example, if M = 64, then $\mathbf{d}$ would have 2,080 exemplars. Given the source domain size $|\mathbf{X}| = 1024 \times 1024$, and 8×8-pixel encoding blocks, the compressed domains of the addend and augend images would each have 16K exemplar indices. Thus, 16K block addition operations are required for compressive addition over $\mathbf{Y}$. However, codebook addition requires only 2,080 block additions, which yields an efficiency of $\eta = 16{,}384/2{,}080 = 7.87$. Thus, parallelism could be reduced nearly eightfold by such VQ-based compressive processing.

11.4.2.4. Example. Consider the VPIC target classification algorithm given in Chapter 10. Here, VPIC exemplar indices are matched to an exemplar-index template. If the template subtends L encoding blocks, then L comparisons and L-1 combination operations (addition, multiplication, or maximum) are required per block position. Thus, given tesselation of an M×N-pixel image into $k \times l$-pixel encoding blocks, a parallel space savings of $CR_D = MN/kl$ is realized, with a computational efficiency of $\eta = \mathbf{O}(CR_D/L)$. However, if the parallel (blockwise) classification operations are a factor of r more efficient than the sequential target recognition operations over each block, then the computational efficiency becomes $\eta = \mathbf{O}(r \cdot CR_D/L)$, and the parallel efficiency $\eta_{||} = \mathbf{O}(r)$, neglecting partitioning overhead.

11.4.2.5. Observation. For purposes of illustration, let us now suppose that the classification algorithm discussed in the preceding section incurs K partitions over a P-processor architecture when processing is conducted over the source domain $\mathbf{X}$. Further assume that an r-fold increase in sequential efficiency is realized per block. Since we can effectively reduce K by a factor of CR_D via compressive processing, it would appear that $\eta_{||} = \mathbf{O}(r \cdot CR_D)$. However, the template matching algorithm requires that L compressed pixels be processed concurrently. Thus, there is an average overlap of $\left\lfloor \left\lceil \sqrt{L} \right\rceil /2 \right\rfloor$ pixels per partition. As a result, the P-pixel partition that was realized through pointwise operations and isomorphic tesselation now incurs an overhead of $2 \cdot \left\lceil \sqrt{L} \right\rceil$ pixels. Given $K' = K/CR_D$ nonoverlapping partitions of $\mathbf{Y}$, each of size P pixels, the overlap requirement of $2 \cdot \left\lceil \sqrt{L} \right\rceil$ pixels per partition will induce $2K' \cdot \left\lceil \sqrt{L} \right\rceil /P$ extra partitions. This possible degradation in efficiency due to partitioning overhead in compressive target recognition is given by

$$\begin{aligned} \Delta\eta &= \frac{K}{K' + 2K' \cdot \left\lceil \sqrt{L} \right\rceil /P} - \frac{K}{K'} \\ &= CR_D \cdot \left(\frac{1}{1 + 2\left\lceil \sqrt{L} \right\rceil /P} - 1 \right). \end{aligned} \tag{415}$$

Note that $\Delta\eta$ is large when L and P are small and CR_D is large. Thus, the maximization of blocksize, which increases CR_D, also tends to decrease P. As a result, there is a design tradeoff between blocksize and compression ratio. Additionally, one must balance template size L with partition size P in order to achieve sufficient resolution (and, therefore, accuracy) in the template representation.

Our current research emphasizes the derivation of optimization expressions for $\Delta\eta$. In future research, we plan to investigate partitioning requirements for a variety of template configurations, including non-rectangular (i.e., L-shaped or serpentine) and sparse templates.

CHAPTER 12
CONCLUSIONS

We present conclusions in Section 12.1 as well as open research issues and suggestions for future work in Section 12.2.

12.1. Conclusions.

This study supports the concept that one can derive operations which simulate image processing operations over the range spaces of various compressive transformations. In many cases of unary pointwise and global reduce operations, the computational speedup equals or exceeds that of the domain compression ratio. However, the speedup obtained for image-template operations varies with the template configuration and image specification. For example, given an uncompressed domain $\mathbf{X}$ and a compressed domain $\mathbf{Y}$, we have shown that morphological erosion and dilation over the range space of one-dimensional Boolean runlength encoded(RLE) imagery requires $\mathbf{O}(|\mathbf{Y}|)$ additions, versus $\mathbf{O}(|\mathbf{X}|)$ multiplications and comparisons. Similar claims hold for edge detection over VPIC-, BTC, and JPEG-compressed 2–D imagery.

In contrast, binary pointwise operations over nonuniformly tesselated real-valued imagery that is compressed blockwise require $\mathbf{O}(|\mathbf{Y}|^2)$ time in the worst case, due to the overhead of $\mathbf{O}(|\mathbf{Y}|)$ search operations incurred per point in $\mathbf{Y}$. The search operations are required to find values in the compressed image that correspond to known points in the uncompressed image, to which the binary operations are referenced. This overhead also pertains to certain image-template operations, especially those with space-variant templates.

However, the preceding disadvantage does not hold universally. Instead, it appears that analytical derivations based upon ϵ-near approximations to transforms or their inverses can

be substituted for the actual transformation over whose range space an analogous or dual image-template operation is computationally burdensome. Using such approximations, we find that:

1. Certain image-template operations (notably, edge detection) can be approximated over a wide range of compression transformations (e.g., BTC, VPIC, JPEG, and RLE) by exploiting inherent properties of the compressed image, versus deriving an edge detection operation in closed form. This is an advantageous result, which incurs only $\mathbf{O}(|\mathbf{Y}|)$ work.
2. Over the range spaces of transformations that preserve edge and boundary information (e.g., RLE and VPIC), component labelling is possible in $\mathbf{O}(|\mathbf{Y}|)$ or $\mathbf{O}(|\mathbf{Y}| \log|\mathbf{Y}|)$ work. This stands in contrast to the known minimum sequential bound of $\mathbf{O}(|\mathbf{X}|)$ work required for sequential connected component labelling of convex components.

Congruent with Cosman's work, we find that the processing of vector-quantized imagery can often be performed efficiently on sequential or parallel architectures using codebook processing only, i.e., by manipulating the codebook exemplars with the source image operation. The chief advantage of this approach is the small amount of work required. For example, if the codebook size is M exemplars, $\mathbf{O}(M)$ work is required, versus $\mathbf{O}(|\mathbf{X}|)$ work for the uncompressed image and a minimum of $\mathbf{O}(|\mathbf{Y}|)$ work for the compressed image. Since M

$$\ll \tag{416}$$

$|\mathbf{Y}|$ is usual, such operations can exhibit computational efficiency that exceeds the compression ratio. Unfortunately, the codebook manipulations required for many arithmetic operations tend to generate a codebook whose statistics poorly characterize the image training set from which the original codebook(s) was(were) developed. As the number of

cascaded operations on the codebook(s) increases, the gradual departure of codebook statistics from image statistics may adversely impact the characterization of other images in the training set by processed codebooks.

A key question of this study pertained to the parallel processing of compressed imagery. Given $CR_d = |\mathbf{X}|/|\mathbf{Y}|$, we have shown that the $\mathbf{O}(|CR_d|)$ efficiencies realized on sequential architectures can, in certain cases, be replicated on parallel architectures, provided that the following constraints are met:

- For optimal efficiency, the image tesselation inherent in the compression or encryption transform should correspond (i.e., be isomorphic to) the multiprocessor indexing scheme. For example, it has been found that SIMD-parallel processor meshes are well-suited to this type of processing, since their Euclidean coordinate system is isomorphic to the rectangular block structure of the encoding block grid employed by many customary compression and encryption transforms.
- The processing elements must be equipped with a set of operators that permits straightforward, efficient implementation of the analogous or dual operations. For example, if an analogous operation requires m-bit addition, that operation should be available on the processor array. Otherwise, synthesis from native operations is required, which new operation may be inefficient. If the synthesized operation is I/O-intensive, then it is often the case that the efficiency of a given parallel analogous or dual operation will decrease per processor. This performance reduction factor generally degrades the performance of parallel compressive processing at least sublinearly, if not linearly.
- Compressive or encryptive processing operations should not require prohibitive I/O overhead to achieve a nominal reduction in computational (versus I/O) operations. Since current parallel architectures have limited interprocessor connectivity, I/O requirements tend to consume processor cycles in work that may not be directly

related to computation. If such I/O cost causes bottlenecks in inter-processor communication (and therefore, in computation), the performance gains that might otherwise be realized by compressive processing will not be apparent.

- When processing encrypted data, interprocessor communication channels (data busses) must be secure, and the data format produced by the encryption transform must not be altered by I/O operations across the processor mesh. For example, in the simple case of transposition ciphers, permutation across the mesh can inadvertently cause total or partial decryption of image neighborhoods. If the processor links are insecure, such partial plaintext can easily be made vulnerable to an adversary.

An additional problem that appears to affect much of practical compressive processing pertains to the accumulation of error in cascaded discrete operations over the range spaces of lossy transforms. Since such transforms have only approximations to their inverses (or, in certain cases, have no inverses), input error (manifested as information loss in the transform) can propagate through the compressive operations, gradually reducing the probability of correctly determining the greylevel of a given pixel. Fortunately, the problem of information loss does not customarily occur in encryptive transforms, due to the fact that cryptosystems are usually isomorphic computational systems. As a result, analogues over the range spaces of encryptive transforms generally incur less information loss than their counterparts over compressive transformations.

However, the preceding comments do not mean that encryptive *computation* is without information loss. In fact, the computational error associated with encryptive processing could, in extreme circumstances, obviate correct decryption. For example, this unpleasant consequence could result from the application of extensively many multi-stage operations (e.g., processing over the range space of DES transforms) when emulating the functionality

of lengthy image processing algorithms. The accrued computational error could, in principle, exceed the precision required for discriminating character indices (i.e., ± 0.5 when the index set is integral). In practice, such problems occur when analog machines (e.g., optical processors) are used to compute cryptographic or cryptanalytic transformations. The instability and error inherent in such processors is well known, and could exhibit a sufficient magnitude to propagate through encryptive processing operations in the aforementioned manner.

As a result of the foregoing observations, we have several suggestions for open research issues and future work.

12.2. Open Issues and Future Work

The following high-level research questions present open issues resulting from the current study:

1. *Are there basic mathematical structures that support a general formulation of image compression?* For example, are there properties of image neighborhoods that facilitate compression by pixel-level processing? In 2–D RLE imagery, we know that a constant-valued region which accommodates a rectangle within its boundary would be preserved. If the region shape was isomorphic to the encoding block shape (customarily rectangular), then such regions would be amenable to compression. If such theory provided useful insights into a unification of image compression operations, then it might be possible to develop a rigorous, predictive technique for better matching compressive transforms to source imagery. Such knowledge would also be useful in deriving efficient compressive operations.
2. *Is the derivation of compressive operations over the range spaces of block encoding transforms (the predominant paradigm) limited to customary (i.e., rectangular) block geometries when the imagery has a Euclidean coordinate system?* Or, are

there other block geometries (i.e., interlocking arbitrarily-shaped blocks) that provide efficient tesselation and processing configurations for certain types of images? This issue pertains directly to image processing over hexagonal lattices, which is useful in the simulation of natural visual systems.

3. *Can variable-blocksize encoding schemes (e.g., fractal-based encoding) be regularized at a resolution level related to a measure derived from the blocksize distribution?* By regularization, we mean converting the variable-blocksize tesselation scheme into a uniform-blocksize tesselation. If this is possible, then the performance of a wide variety of pointwise and image-template operations over RLE- and IFS (fractal)-compressed imagery could be drastically improved. Whereas the computational cost is currently $\mathbf{O}(|\mathbf{Y}|^2)$, it would be possible to achieve $\mathbf{O}(|\mathbf{Y}|)$ cost, albeit with additional error due to approximations involved (for example) in grouping small blocks into larger blocks.

4. *Are there novel methods for image coding that minimize (or optimize with respect to operational constraints) the propagation of lossy transformation error through various compressive operations?* If so, what classes or subclasses of our transform taxonomy appear to provide improved results? This concept would be especially useful in the processing of radiological imagery (e.g., detection of small precancerous lesions in mammograms or MRI scans). In such cases, accuracy of object location can be constrained to pixel or sub-pixel resolution. However, image display errors alone often obviate such discrimination, which errors can increase when computational error due to image processing is factored into the error budget. Hence, without improved error prediction methods, this remains an unsettled question.

5. *What interprocessor connection schemes tend to render encrypted imagery more vulnerable, due to the inadvertent inversion of transpositions that may be effected*

by the encryption transform? This issue will become more important as interprocessor connectivity schemes become more flexible and such connectivity increases. For example, in a butterfly (FFT) processor, the generation of permutation sequences is nearly trivial. Could the presence of such permutations obviate encryptive processing on highly connected processor networks? We plan to address combinatorial aspects of this question in future research.

Issue 1), the elucidation of mathematically tractable, unifying structure for image compression, is analogous to much ongoing work in the mathematics of encryption. Although the latter has been well supported by the defense community, until recently, fundamental research in image compression has considerably lagged encryption research. With the increasing use of wide-area networks for image transmission, compression theory is receiving more attention in the literature. Nevertheless, current trends in development appear to support the determination of minimum image information content, rather than theory that would predict image compressibility in relation to physical constraints (e.g., visual picture quality).

Ongoing and future work in the area of processing compressed imagery is expected to emphasize the following practical issues:

- *Implementation of compressive processing algorithms that were developed in this study on the PAL-I and PAL-II architectures.* PAL is a novel, high-bandwidth SIMD-parallel architecture that is based upon Lockheed-Martin's GAPP-IV processor and directly implements a large subset of nonrecursive image algebra. Although PAL was scheduled to be available for performance testing during the term of this study, various delays in hardware and software development made PAL implementation infeasible within the allotted time. Thus, we expect to begin early PAL implementations in late 1996.

- *Further computation of error figures and simulation of error propagation in compressive processing.* Apart from the issue of rendering image-template operations more efficient, the adverse effects of error propagation through lossy transforms and cascaded compressive operations remains the chief disadvantage of compressive processing. This effect must be researched in detail at various resolution and computational precision levels, in order to determine limits of detectability for known objects in compressively processed imagery. With such results, we could better determine the feasibility of compressive processing for on-board target recognition applications using high-bandwidth processors (such as PAL) at high frame rates.
- *Investigation of template decomposition procedures for more efficient compressive image-template operations.* The difficulty with which complicated space-variant template configurations are implemented over the range space of certain compressive transforms leads naturally to the conjecture that compressive processing could benefit from the large amount of work performed in template decomposition. This appears to be particularly true in the case of template decompositions for block-encoded imagery, due to problems associated with representing template blocks in terms of complete encoding blocks. Such representational problems are especially crucial for template specification in target recognition applications, where the preservation of template information can be crucial to the compressive detection partially resolved targets (several pixels to hundreds of pixels on target).

JPEG compressive operations present many unresolved issues. For example, computation of the quality and smoothing parameters varies with JPEG implementation and is occasionally proprietary. How are such parameters related to the information content of the DCT coefficients that are retained/discarded? What, if any, spatial information (e.g., source image MTF) is accounted for in such measures? (The MTF is useful as a measure of spatial frequencies discarded by the DCT coefficient quantization step.) Additionally, the

computation of JPEG's Huffman codebook has not been extensively reported for commercial JPEG implementations. Although we do not consider the Huffman encoding aspect of JPEG in this study, it is certainly important to know how such encoding occurs in a practical implementation, in order to access the reduced matrix of DCT coefficients. Thus, we plan to survey JPEG algorithms and their manufacturers to obtain several commercial JPEG algorithms whose output can be inverted piecewise.

We also plan to investigate information loss associated with VPIC edge detection and morphological operations for a variety of textual (Boolean) or natural imagery. The results of such research are expected to provide useful information for the design of improved VPIC edge and erosion/dilation codebooks. Based upon our current research results, we further expect that an in-depth understanding of VPIC's error behavior for simple (i.e., von Neumann) edge detection and morphological structuring elements would facilitate further improvements in VPIC morphological operations with multi-pixel and non-von Neumann templates.

The issue of error statistics is important and cannot be overlooked (or overstressed) in ATR applications. This issue is of particular interest when the ATR algorithm perturbs the input noise distribution in an unexpected manner. For example, we have presented various error figures for VPIC, JPEG, and image rotation (Chapter 8). In each case, although the input image was generally perturbed with Gaussian noise produced by the imaging system, the error statistics were markedly non-Gaussian, and included multimodal distributions. We are interested in determining not only the magnitude and probability of such errors, but also the form of their distributions, where possible.

From such research, we eventually hope to gain a better (and possibly, an analytical or formulaic) understanding of how error propagates through ATR algorithms and their compressive analogues. For example, consider that the gradient magnitude distribution of a natural image is frequently Lorentzian [86]. Regardless of whether the image is perturbed with Poisson or Gaussian noise, the gradient distribution remains Lorentzian. The reasons

for this behavior are not apparent. From the definition of an analogue, we expect that such effects are produced by a given compressive analogue of a gradient operation. However, comprehensive testing of the preceding hypothesis remains to be completed. Now, suppose we perturb the analogous gradient operation such that significant computational error (approximating the input noise magnitude) is introduced into the gradient output. Will the gradient distribution remain Lorentzian? Such questions have not been answered in the literature, and represent a large, open area of research in the statistics of image processing that would have crucial impact on error-dependent processes. Medical image understanding (i.e., recognition of known tumor types) and ATR of small targets represent key applications of such analyses, and could benefit from improvements in computational accuracy. The results of such research could enhance and expand analytical knowledge that is often lacking when error analyses of complex algorithms are attempted.

REFERENCES

[1] Cosman, P.C., K.L. Oehler, E.A. Riskin, and R. Gray. "Using vector quantization for image processing", *Proceedings of the IEEE* **81**:1325–1341 (1993).

[2] Schmalz, M.S. "General theory for the processing of compressed and encrypted imagery, with taxonomic analysis",*Proceedings SPIE* **1702**:250-263 (1992).

[3] Schmalz, M.S. "On the processing of compressed and encrypted imagery: Complexity analyses, with application to novel regimes of efficient computation", in *Proceedings SPIE* **1700** (1992).

[4] Schmalz, M.S. "On the processing of compressed data: Application to morphological and grey-scale computations over runlength-, block-, transform-, and derivative-encoded imagery",*Proceedings SPIE* **1702**:60-75 (1992).

[5] Schmalz, M.S. "On the processing of runlength-encoded Boolean imagery", in *Proceedings SPIE* **1955** (1993).

[6] Schmalz, M.S. "On the parallel processing of runlength-encoded Boolean imagery", in *Proceedings SPIE* **1955** (1993).

[7] Leger, A., T. Omachi, and G.K. Wallace. "JPEG still picture compression algorithm", *Optical Engineering* **30**:947–954 (1991).

[8] Modestino, J.W. and Y.H. Kim. "Adaptive entropy-coded predictive vector quantization of images", *IEEE Transactions on Signal Processing* **40**:633–644 (1992).

[9] Chen, D. and A.C. Bovik. "Visual pattern image coding",*IEEE Transactions on Communications* **38**:2137–2146 (1990).

[10] Duff, I.S., R.G. Grimes, and J.G. Lewis. "Sparse matrix test problems", *ACM Transactions on Mathematical Software* **15**:1–14 (1989).

[11] Liu, J.W.H. "The multifrontal method for sparse matrix solution: theory and practice", *SIAM Review* **34**:82–109 (1992).

[12] Oehler, K.L., P.C. Cosman, R.M. Gray, and J. May. "Classification using vector quantization", in *Proceedings of the 25th Annual Asilomar Conference on Signals, Systems, and Computers* (Pacific Grove, CA, Nov. 1991), pp. 439–445 (1991).

[13] Netravali, A.N. and J.O Limb. "Picture coding: A review", *Proceedings of the IEEE* **68**:366-406 (1980).

[14] Zschunke, W. "DPCM picture coding with adaptive prediction", *IEEE Transactions on Communication*, **COM-25**(11):1295-1302 (1977).

[15] Li, W. and Y.Q. Zhang. "Vector-based signal processing and quantization for image and video compression", *Proceedings of the IEEEE* **83**:317–335 (1995).

[16] Healy, D.J. and O.R. Mitchell. "Digital video bandwidth compression using block truncation encoding", *IEEE Transactions on Communication* **COM-29**(12):1809–1817 (1981).

[17] Kunt, M. M. Bernard, and R. Leonardi. "Recent results in high-compression image coding", *IEEE Transactions on Circuits and Systems* **CAS-34**(11):1306-36 (1987).

[18] Baron, S. and W.R. Wilson. "MPEG overview", *SMPTE Journal* **103**:391–394 (1994).

[19] Felician, L. and A. Gentili. "A nearly-optimal Huffman encoding technique in the microcomputer environment", *Information Systems* (UK) **12**(4):371-3 (1987).

[20] Shoaff, W.D. "The Singular Value Decomposition Implemented in Image Algebra", M.S. Thesis, Department of Computer and Information Sciences, University of Florida (1986).

[21] Ohta, Mutsumi, and S. Nogaki. "Hybrid picture coding with wavelet transform and overlapped motion-compensated interframe prediction coding", *IEEE Transactions on Signal Processing* **41**:3416–3424 (1993).

[22] Dettmer, R. "Form and function: Fractal-based image compression", *IEE Review* **38**:323–327 (1992).

[23] Bani-Eqbal, B. "Enhancing the speed of fractal image compression", *Optical Engineering* **34**:1705–1710 (1995).

[24] Mandelbrot, B.B. *The Fractal Geometry of Nature*, 2nd Edition, New York: W.H. Freeman (1983).

[25] Barlow, H.B., and P. Fatt, Eds. *Vertebrate Photoreception*, New York: Academic Press ((1976).

[26] Kahn, D. *The Codebreakers*, New York: Macmillan (1967).

[27] Sinkov, A. *Elementary Cryptanalysis: A Mathematical Approach*, Washington, DC: Mathematical Association of America (1966).

[28] Koblitz, N. *A Course in Number Theory and Cryptography*, 2nd Edition, New York: Springer-Verlag (1994).

[29] Meyer, C.H. *Cryptography: A New Dimension in Computer Security*, New York: Wiley (1982).

[30] Patterson, W. *Mathematical Cryptology for Computer Scientists and Mathematicians*, Totowa, NJ: Rowman and Littlefield (1987).

[31] Nechtaval, J. *Public-Key Cryptography*, Gaithersburg, MD: US Department of Commerce, National Institute of Standards and Technology (1991).

[32] Fisher, D. and I. Bahl, Editors. *Gallium Arsenide IC Applications Handbook*, San Diego: Academic Press (1995).

[33] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading,MA:Addison-Wesley (1989).

[34] van der Bank, D.J. "The Use of Genetic Algorithms for Cryptanalysis", PhD Dissertation, University of Pretoria, SA (1992).

[35] Karnin, E.D. "Probabilistic and Computational Methods in Cryptography", PhD Dissertation, Stanford University (1984).

[36] Chor, B.Z. *Two Issues in Public-Key Cryptography: RSA Bit Security and a New Knapsack Type System*, Cambridge, MA: MIT Press (1986).

[37] Cobb, S.E. *Public-Key Cryptography and the Zero-One Knapsack Problem*, M.Sc. Thesis, Carleton University, Canada (1983).

[38] Jan, J.K. and S.J. Wang. "A dynamic access control scheme based upon the Knapsack problem", *Computers and Mathematics with Applications* **26**:75–86 (1993).

[39] Amirazizi, H.R. "Time-Memory Processor Tradeoffs (Cryptography, Multiple-Encryption, Knapsack Problem)", Ph.D. Dissertation, Stanford University (1986).

[40] Evaluated in Hartman, W.J. "A Critique of Some Public-Key Cryptosystems", Report 126–D-3, US. Department of Commerce, National Telecommunications and Information Administration (1980).

[41] Takagi, N. "A radix-4 modular multiplication hardware algorithm for modular exponentiation", *IEEE Transactions on Computers* **41**:949–956 (1992).

[42] Rivest, R.L., L. Adelman, and M.L. Dertouzos. "On data banks and privacy homomorphisms", in *Foundations of Secure Computations*, R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton, Eds., New York: Academic Press, pp. 169–179 (1978).

[43] Abadi, M., J. Feigenbaum, and J. Kilian. "On hiding information from an oracle", *Journal of Computer System Science* **39**:21–50 (1989).

[44] Ahituv, N., Y. Lapid, and S. Neumann. "Processing encrypted data", *Communications of the ACM* **30**:777–779 (1987).

[45] Yu, K.W. and T.L. Yu. "Superimposing encrypted data", *Communications of the ACM* **34**:49–54 (1991).

[46] Nechtaval, J. *Public-Key Cryptography*, NIST Special Publication **800–2**, Gaithersburg, MD: Department of Commerce (1991).

[47] Jawerth, B. and W. Sweldens. "An overview of wavelet based multiresolution analyses", *SIAM Review* **36**:377–412 (1994).

[48] Jolion, J.M. and A. Montanvert. "The adaptive pyramid: A framework for 2D image analysis", *CVGIP: Image Understanding* **55**:339–348 (1992).

[49] Healy, D.J. [On edge detection over locally avereaged images], Unpublished report for AFOSR Summer Research Grant (1978).

[50] Evaluated in Burt, P.J. and E.H. Adelson. "The Laplacian pyramid as a compact image code", *IEEE Transactions on Communications* **COM-31**:532-540, (1983).

[51] Schmalz, M.S. "On the processing of compressed imagery. 2. Compressive operations with VPIC-, BTC-, VQ-, and JPEG-compressed imagery", in *Proceedings SPIE* **2751** (1996).

[52] Schmalz, M.S. "Automated target recognition in compressed stereo images", to appear in *Proceedings SPIE 1996 Technical Symposium, Denver, CO, August 1996.*

[53] Schmalz, M.S. "Automated detection and recognition of small targets in compressed imagery. 1. Background and theory", in *Proceedings SPIE* **2765** (1996).

[54] Ritter, G.X. and J.N. Wilson. *Handbook of Image Processing Algorithms in Image Algebra*, Boca Raton, FL: CRC Press (1996).

[55] Ritter, G.X. *Image Algebra* (In preparation, partial manuscript avaialble via anonymous FTP through jnw@cis.ufl.edu).

[56] Schmalz, M.S. and G.X. Ritter. "Image-algebraic design of multispectral target recognition algorithms", in *Proceedings SPIE* **2300**:213–228 (1994).

[57] Li, D. "Recursive Operations in the Image Algebra and their Applications to Image Processing", Ph.D. Dissertation, Department of Computer and Information Sciences, University of Florida, Gainesville, FL (1990).

[58] Koza, J. *Genetic Programming II: Automatic Discovery of Reusable Programs*, Cambridge, MA: MIT Press (1994).

[59] National Research Council (USA). *Computers at Risk: Safe Computing in the Information Age*, Washington, DC: National Academy Press (1991).

[60] Bevington, P.R. *Data Reduction and Analysis for the Physical Sciences*, New York: McGraw-Hill (1969).

[61] Clifford, A.A. *Multivariate Error Analysis: A Handbook of Error Propagation and Calculation in Many-Parameter Systems*, London: Applied Science Publishers (1973).

[62] Schmalz, M.S., G.X. Ritter, C.T. Yang, and W.C. Hu. "Center-surround filters for the detection of small targets in cluttered multispectral imagery. 2. Analysis of errors and filter performance", in *Proceedings SPIE* **2496** (1995).

[63] Sigelman, J. *Retinal Diseases: Pathogenesis, Laser Therapy, and Surgery*, Boston, MA: Little, Brown (1984).

[64] Chern, W.S. "Error Propagation in Discrete Processes", PhD Dissertation, Department of Electrical Engineering, University of Florida, Gainesville, FL (1970).

[65] Jiang, L.M., T.F. Lee, and T.T. Hwang. "Performance-driven interconnection optimization for microarchitecture synthesis", *IEEE Transactions on Computer-Aided Deignh of Integrated Circuits and Systems* **13**:137–149 (1994).

[66] Pardalos, P.M., Ed. *Advances in Optimization and Parallel Computing*, New York: North-Holland (1992).

[67] Kim, W.Y. "3–D object recognition using bipartite matching embedded in discrete relaxation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**:224–251 (1994).

[68] Shannon, C. "Communication theory of secrecy systems", *Bell System Technical Journal* **28**:656–715 (1949).

[69] Ash, R.B. *Information Theory*, New York: Interscience (1965).

[70] Abramson, N. *Information Theory and Coding*, New York: McGraw-Hill (1963).

[71] Kersten, D. "Predictability and redundancy of natural images", *Journal of the Optical Society of America-A* 4:2395–2400 (1987).

[72] Akl, S. *The Design and Analysis of Parallel Algorithms*, Englewood Cliffs, NJ: Prentice Hall (1989).

[73] Silvera, E. "Adaptive pattern recognition with rotation, scale, and shift invariance", *Applied Optics* **34**:1891–1900 (1995).

[74] Cornsweet, T. *Visual Perception*, New York: Academic Press (1970).

[75] Zrenner, E. *Physiology of the Human Eye and Visual System*, Hagerstown, MD: Harper&Row (1979).

[76] Haralick, R., K. Shanmugan, and I. Dinstein. "Textural features for image classification", *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-3**:610–621 (1973).

[77] Haralick, R. "Statistical and structural approaches to textures", *Proceedings IEEE* **67**:786–804 (1979).

[78] Weszka, J., C. Dyer, and A. Rosenfeld. "A comparative study of texture measures for terrain classification", *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-6**:269–285 (1976).

[79] Pal, N.R., J.C. Bezdek, and E.C.K. Tsao. "Generalized clustering networks and Kohonen's self-organizing scheme", *IEEE Transactions on Neural Networks* 4:549–547 (1993).

[80] Kim, Y.K. and J.B. Ra. "Adaptive learning method in self-organizing map for edge preserving vector quantization", *IEEE Transactions on Neural Networks* **6**:278–280 (1995).

[81] Yang, C., W.C. Hu, M.S. Schmalz, and G.X. Ritter. "Center-surround filters for the detection of small targets in cluttered multispectral imagery. 1. Background and filter design", in *Proceedings SPIE* **2496** (1995).

[82] Prewitt, J.M.S. "Object enhancement and extraction" in B.S. Lipkin and A. Rosenfeld, Eds., *Picture Processing and Psychopictorics*, New York: Academic Press (1970).

[83] Sobel, I.E. *Camera Models and Machine Perception*, Ph.D. Dissertation, Stanford University, Stanford, CA (1970).

[84] Kirsch, R.A. "Computer determination of the constituent structure of biological images", *Computers and Biomedical Research* 4(3):315–328 (1971).

[85] Gonzalez, R.C. and P. Wintz. *Digital Image Processing*, Reading, MA: Addison-Wesley (1987).

[86] Lettington, A. and Q.H. Hong. "Interpolator for infrared images", *Optical Engineering* **33**:725–729 (1994).

BIOGRAPHICAL SKETCH

Mark Schmalz was awarded the baccalaureate degree in applied physics (minor in electrical engineering) from Michigan Technological University in 1974. Following conferral of the O.D. degree in optometry (minor in optics) from Pacific University in 1979, Dr. Schmalz maintained an active clinical practice from 1980 to 1984. From 1982 through 1988, the author conducted research at the Harbor Branch Foundation and the Harbor Branch Oceanographic Institution (Florida). Since 1989, he has conducted research at the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, where he received the M.S. in Computer and Information Sciences in 1991. Dr. Schmalz has published over 60 research papers in automated target recognition, optical propagation theory, image compression and restoration, parallel computation, and cryptography.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Gerhard X. Ritter, Chairman
Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Joseph N. Wilson
Assistant Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Andrew F. Laine
Associate Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Sanguthevar Rajasekaran
Associate Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John Harris
Associate Professor of
Electrical and Computer Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 1996

Winfred M.Phillips
Dean, College of Engineering

Karen M. Holbrook
Dean, Graduate School

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Gerhard X. Ritter, Chairman
Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Joseph N. Wilson
Assistant Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Andrew F. Laine
Associate Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Sanguthevar Rajasekaran
Associate Professor of
Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John Harris
Associate Professor of
Electrical and Computer Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 1996

Winfred M.Phillips
Dean, College of Engineering

Karen M. Holbrook
Dean, Graduate School